

МІНІСТЕРСТВО ОСВІТИ І НАУКИ
УКРАЇНИ
Тернопільський державний технічний
університет імені Івана Пулюя

Кафедра автоматизації
технологічних процесів
і виробництв

КОНСПЕКТ ЛЕКЦІЙ

з курсу

”Мікропроцесорні та програмні засоби
автоматизації”

Тернопіль 2009

Посібник розроблений відповідно до навчальної програми курсу "Мікропроцесорні та програмні засоби автоматизації".

Укладачі: Медвідь В. Р. доц., к.т.н., Козбур В.Р. асистент, Пісьціо В.П. асистент

Рецензенти:

Стухляк П.Д завкафедри. "Комп'ютерноінтегрованих технологій", д.т.н., проф.;
Карпінський М.П завкафедри. "Комп'ютерної інженерії" ТНЕУ, д.т.н., проф.

Розглянуто й затверджено на засіданні кафедри автоматизації технологічних процесів та виробництв (протокол № 9 від 12 лютого 2008 р.)

Схвалила й рекомендовала до друку вчена рада факультету комп'ютерних технологій Тернопільського державного технічного університету імені Івана Пулюя (протокол № 5 від 22 лютого 2008 р.).

Зміст

Мікропроцесор i80286	8
1. Основні особливості архітектури мікропроцесора i80286	8
1.1. Загальні характеристики структури МП	8
1.2. Інтерфейс процесора i80286	9
2. Регістри МП i80286	15
3. Переривання у МП i80286	19
4. Реальний режим роботи	22
5. Захищений режим роботи	25
5.1. Регістри дескрипторної таблиці	27
5.2. Дескриптори	27
5.3. Сегмент стану TSS та переключення задач	30
5.4. Захист у процесорі i80286	32
5.4.1. Рівні захисту	32
5.4.2. Межі сегментів	34
5.4.3. Привілейовані команди	34
5.5. Міжсегментні переходи	36
5.6. Віртуальна пам'ять	38
Контрольні питання	39
Мікропроцесор i386	40
1. Основні характеристики	40
2. Структура мікропроцесора i386	41
3. Призначення виводів мікропроцесора	44
4. Регістри i386	45
5. Переривання	52
6. Реальний режим роботи	56
7. Захищений режим	59
7.1. Сегменти	60
7.2. Селектори	60
7.3. Дескрипторні таблиці	61

7.4. Дескриптори	62
7.5. Формування лінійної адреси	66
7.6. Сторінки	68
8. Захист	72
8.1. Привілеї	73
8.2. Використання привілейованих команд	74
8.3. Ввід-вивід	74
8.4. Захист на рівні сегментів	75
8.5. Захист на рівні сторінок	76
8.6. Захист при передачі керування. Шлюзи. Міжсегментні переходи	77
8.7. Режим віртуального 86	79
9. Віртуальна пам'ять	80
9.1. Віртуальна пам'ять на рівні сегментів	81
9.2. Віртуальна пам'ять на рівні сторінок	82
10. Перемикання задач	84
10.1. Регістр TR	84
10.2. Сегмент стану задачі	85
10.3. Зміна задачі	87
11. Система команд i386	87
11.1. Основні типи даних МП	87
11.2. Загальний формат команди МП	88
11.3. Команди пересилання даних	90
11.4. Арифметичні команди	92
11.5. Команди обробки бітів	94
11.6. Команди обробки стрічок	97
11.7. Команди передавання керування та організації циклів	98
11.8. Команди керування МП	101
11.9. Команди керування захистом	102
Контрольні питання	104
Мікропроцесор i486	105

1. Особливості організації та основні характеристики	105
2. Формат регістра CR0 i486	106
3. Формат регістра CR3 процесора i486	107
4. Системні команди мікропроцесорів i80286/i386/i486	108
Архітектура процесорів Pentium	111
1. Особливості організації МП Pentium	111
2. Системна шина процесорів Pentium	114
Мікро-ЕОМ сімейства МК48	120
1. Сімейство однокристальних мікро-ЕОМ	120
2. Призначення виводів мікросхеми	121
3. Основні функціональні блоки ОМЕОМ	124
3.1. Порти	128
3.2. Лічильник-таймер	130
3.3. Переривання	131
4. Система команд МК48	132
4.1. Загальні відомості про систему команд	132
4.2. Класифікація команд МК48	133
4.3. Способи адресації даних	133
4.4. Модифікація РССП та ознак користувача	133
4.5. Група команд пересилання даних	134
4.6. Група команд арифметичні операції	137
4.7. Група команд роботи з бітами	138
4.8. Група команд передавання керування	140
4.9. Команди керування режимом роботи лічильника-таймера	144
4.10. Команди перемикання банків регістрів і та пам'яті програм	145
5. Програмування ОМЕОМ	146
5.1. Приклади використання команд передачі даних	146
5.2. Приклад використання команд арифметичних операцій	148
5.3. Приклади використання команд логічних операцій	152
5.4. Приклади використання команд передавання керування і команд	

керування режимом МК48	155
Контрольні запитання	158
Архітектура однокристальних мікро-ЕОМ сімейства МК-51	160
1. Однокристальні мікро-ЕОМ сімейства МК-51	160
2. Архітектура ОМЕОМ і80С51	163
3. Регістр ознак	167
4. Умовне позначення мікросхеми	168
5. Мінімальна схема включення	171
6. Загальна структура пам'яті ОМЕОМ і80С51	171
7. Структура внутрішньої пам'яті даних	173
8. Система команд ОМЕОМ МК-51	176
8.1. Група команд пересилання даних	177
8.2. Група команд арифметичних операцій	179
8.3. Група команд логічних операцій	183
8.4. Група команд операцій з бітами	184
8.5. Група команд передачі керування	185
8.6. Команди операцій зсуву	189
8.7. Інші команди	190
9. Паралельні порти вводу-виводу інформації	190
10. Система переривань	193
11. Лічильники-таймери	199
12. Послідовний порт	204
13. Режим зниженого енергоспоживання	208
Контрольні запитання	209
Мікроконтролери PIC16C5X та PIC16C8X	210
1. Контролер PIC16C84. Організація, програмування та застосування	215
1.1. Відмінності PIC16C84 від PIC16C5x	218
1.2. Перехід від PIC16C5x до PIC16C84	219
2. Умовне позначення та призначення виводів мікросхеми	219
3. Структурна схема PIC16C84	221

4. Регістр STATUS	222
5. Структура пам'яті мікроконтролера	224
5.1. Організація вбудованого ПЗП	224
5.2. Лічильник команд та стек	224
5.3. Структура ОЗП мікроконтролера	226
5.4. EEPROM даних	228
6. Порти	230
7. Таймери	233
7.1. Лічильник таймер RTCC	233
7.2. Watchdog-таймер	236
7.3. Попередній подільник	237
8. Переривання	238
9. Система команд мікроконтролера	240
9.1. Команди роботи з байтами	241
9.2. Команди роботи з бітами	243
9.3. Команди переходів	243
9.4. Спеціальні команди	244
10. Режим зниженого енергоспоживання	244
11. Скидання та початкове встановлення	245
11.1. Джерела скидання	245
11.2. Скидання при відкненні живлення	247
11.3. Watcdog-таймер	248
12. Генератор та синхронізація	248
12.1. Типи генераторів	248
12.2. Робота з кварцовим резонатором	249
12.3. Синхронізація від зовнішніх джерел	249
12.4. RC-генератор	250
13. Конфігурація та захист	251
Контрольні запитання	252
Список літератури	253

Мікропроцесор i80286

Фірма Intel в 1982 р. оголосила про випуск мікропроцесора i80286. Цей пристрій зберігає сумісність з мікропроцесором i8086, як і мікропроцесор i80186, але значно відрізняється від нього своїми можливостями завдяки додатковим засобам захисту. У мікропроцесорі i80286 немає внутрішніх генератора синхронізації, логіки вибору мікросхем, каналів DMA, таймерів і контролера переривань. Замість цього є засоби керування пам'яттю і віртуальною пам'яттю.

1. Основні особливості архітектури мікропроцесора i80286

1.1. Загальні характеристики структури МП

Мікропроцесор i80286 є другим поколінням 16-розрядних мікропроцесорів. Його особливості:

- ◇ тактова частота від 8 до 10 МГц, продуктивність у 2-3 рази вища, ніж у i8086;
- ◇ адресує до 16 Мбайт фізичної пам'яті, з можливістю виділення кожній програмі віртуальної пам'яті розміром до 1 Гбайта;
- ◇ реалізовано конвеєрне виконання команд;
- ◇ реалізовано багатозадачний режим роботи.

Найістотнішою відмінністю від i8086 є механізм керування адресацією пам'яті, що забезпечує 4-рівневу систему захисту і підтримання віртуальної пам'яті, та спеціальні засоби призначені для підтримання механізму перемикання задач.

Мікропроцесор має розширену систему команд, яка містить усі команди i8086 (крім "напівлегальної" команди POP CS), кілька нових команд загального призначення та команди керування захистом. Процесор i80286 може працювати в двох режимах роботи: реальному, котрий є повністю сумісним з i8086, та захищеному. Мікропроцесор i80286 має 16-бітну шину даних і 24 розрядну адресну шину з котрої в реальному режимі використовуються лише 20 розрядів.

У реальному режимі можлива адресація до 1 Мбайта фізичної пам'яті. За

складом і призначенням у реальному режимі регістри i80286 в основному, збігаються з регістрами i8086. Зміни стосуються призначення бітів регістра ознак і можливістю переключення у захищений режим. У процесі виконання програми МП використовує 6-байтну чергу команд. Відзначимо, що завдяки архітектурі скорочено час виконання команд, тому підпрограми часової затримки на цьому МП працюють швидше ніж на i8086.

У захищеному режимі мікропроцесор дозволяє адресувати до 16 Мбайтів фізичної пам'яті, на які для кожної задачі з використанням механізму сегментної адресації та зовнішніх регістрів сторінок можуть відображатися до 1 Гбайта віртуальної пам'яті. Переключення у захищений режим здійснюють однією командою (з попередньо підготовленими таблицями дескрипторів). Зворотнє переключення у реальний режим можливе тільки через апаратне скидання мікропроцесора, що вимагає відносно значного часу.

1.2. Інтерфейс процесора i80286

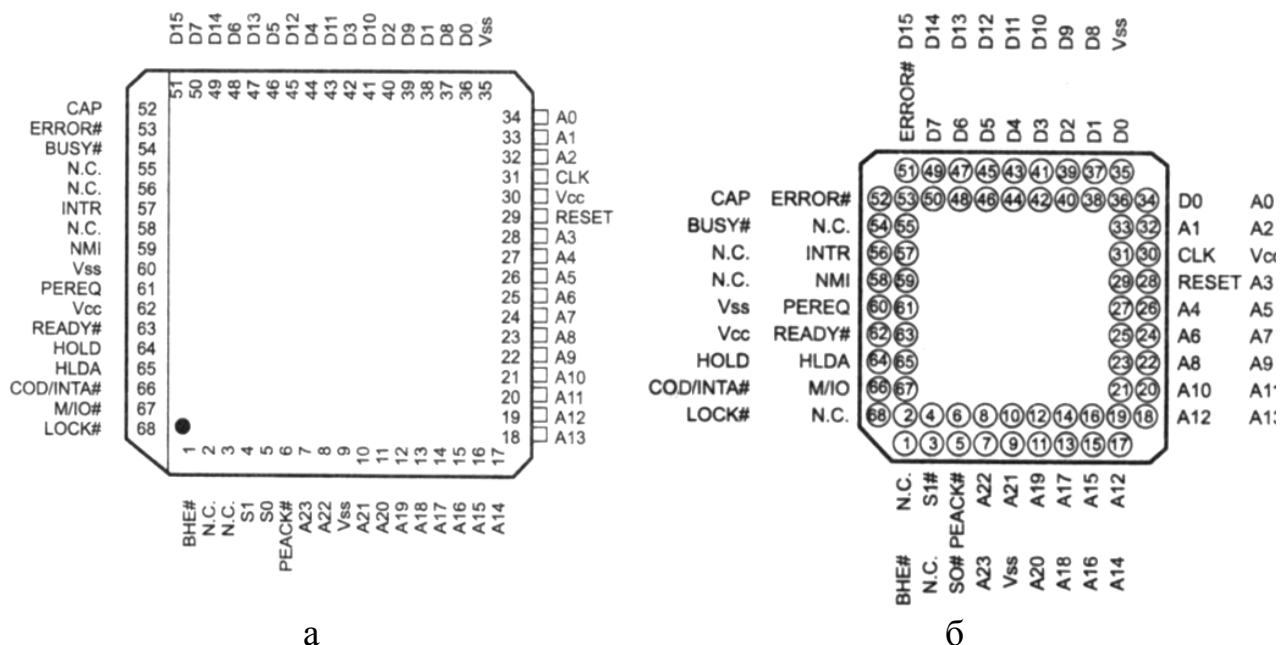


Рис. 1. Розташування виводів процесора i80286

а - в корпусі PLCC, б – в корпусі PGA

Процесори i80286 випускали в 68-вивідних корпусах PLCC і PGA (рис.1). Їх інтерфейс відрізняється від i8086 використанням розділних шин адрес і даних, конвеєрної адресації, а також складом та призначенням керуючих

сигналів (таблиця 1), передбачено можливість використання математичного співпроцесора i80287.

Таблиця 1. Призначення сигналів процесора 80286

Сигнал	Тип виводу	Призначення
1	2	3
CLK	Вхід	Clock – зовнішній сигнал синхронізації процесора
D[0:15]	Вхід-вихід	Data – сигнали шини даних. У персональному комп'ютері (PC) ці сигнали через буферні регістри з'єднуються з відповідними лініями Data[0:15] шини ISA, сигнали D[8:15] через додатковий буфер з'єднуються з лініями Data[0:7] для передавання байта з непарною адресою
A[0:23]	Вихід	Address – сигнали шини адреси. У PC сигнали A[0:19] через регістр виводяться на лінії Addr[0:19] шини ISA. Сигнали A[17:23] надходять на лінії LA[17:23] без запам'ятовування
BHE#	Вихід	Byte High Enable. Сигнал BHE# вказує на присутність даних на лініях AD[8:15]. У PC через регістр подається на лінію SBHE# системної шини
S0, S1	Вихід	Bus Status – сигнали стану, спільно з M/IO# і COD/INTA# визначають тип шинного циклу. Сигнали надходять на входи контролера шини (i82288), який їх декодує у сигнали керування системної шини IOR#, IOW#, MEMR#, MEMWR#, INTA# і ALE
M/IO	Вихід	Memory/IO Select – високий рівень вказує на передавання даних до пам'яті (чи зупинку МП), низький – на передавання даних до портів вводу-виводу або підтвердження переривання
COD/INTA#	Вихід	Code/Interrupt Acknowledge – вказує на вибір коду при читанні пам'яті або цикл підтвердження переривання
LOCK#	Вихід	Bus Lock – сигнал монополізації керування шиною, виробляється на час виконання команди по префіксу інструкції LOCK і автоматично при виконанні інструкції XCHG з пам'яттю, підтвердження переривання і доступу до таблиць дескрипторів
READY#	Вхід	Bus Ready — сигнал завершення шинного циклу. Шинний цикл триває до появи сигналу готовності (низький рівень). Синхронізований сигнал готовності потрапляє через мікросхему i8284 з лінії IOCHRDY системної шини PC і логіки керування тривалістю циклів апаратів системної плати
RESET	Вхід	Сигнал апаратного скидання (високим рівнем)

1	2	3
INTR	Вхід	Interrupt Request — сигнал запиту (високий рівень) маскованого переривання. У PC потрапляє з виходу контролера переривань i8259A
NMI	Вхід	Non Mascable Interrupt — сигнал, високий рівень якого викликає немасковане переривання NMI.
HOLD HLDA	Вхід-вихід	Bus Hold Request і Hold Acknowledge – запит керування локальною шиною й підтвердження надання шини, відповідно. Активний рівень кожного з сигналів – високий
PEREQ PEACK#	Вхід-вихід	Processor Extension Operand Request і Acknowledge – запит на передавання операндів співпроцесору і підтвердження його завершення
BUSY# ERROR#	Вхід Вхід	Processor Extension Busy і Error – сигнали стана співпроцесора. Активність сигналу BUSY# зупиняє процесор, а сигнал ERROR викликає виключення при виконанні інструкцій WATT і деяких інструкцій ESCAPE
CAP	–	Substrat Filter Capasitor – вивід для під'єднання зовнішнього конденсатора, потрібного для внутрішнього подільника частоти 80286. Для КМОП-процесора 80C286 конденсатор не потрібен

Таблиця 2. Декодування стану процесора 80286

COD/ INTA#	M/IO#	SI#	S0#	Тип циклу
0	0	0	0	Підтвердження переривання (INTA#)
0	1	0	0	Якщо A0=1 – то МП зупинений, у противному випадку відключення (Shutdown)
0	1	0	1	Читання даних з пам'яті (MEMRD#)
0	1	1	0	Запис у пам'ять (MEMWR#)
1	1	0	1	Вибірка коду (MEMRD#)
1	0	0	1	Читання порту (IORD#)
1	0	1	0	Запис у порт(IOWR#)
0	0	0	1	Не використовується
0	0	1	0	
1	1	x	0	
1	x	0	0	
x	x	1	1	Пасивний стан шини
x – будь-який стан лінії				

Шина адреси дозволяє адресувати 16 Мбайт фізичної пам'яті в захищеному режимі і 1 Мбайт із області молодших адресів — у реальному

режимі. Із цього правила для реального режиму є два винятки:

- ◇ лінії A[20:23] після сигналу RESET на час вибірки коду перебувають в одиничному стані до першої інструкції міжсегментного переходу;
- ◇ лінія A20 може приймати одиничні значення, коли поєднання значень сегментного регістра і ефективної адреси призводить до переходу через межу 0FFFFFFh (максимально можливі значення фізичної адреси в реальному режимі 10FFFEh). Для забезпечення повної програмної сумісності з i8086 застосовують програмно-керований вентиль, що примусово обнулює лінію A20 системної шини адреси.

При адресації портів вводу-виводу процесор використовує тільки біти A[0:15] шини адреси, старші розряди шини адреси переводяться у стан лог. 0.

Як і в процесорі i8086 обмін по шині даних можливий байтами або словами. При побайтовому обміні байт з парною адресою (A0=0) передається по лініях D[0:7], з непарною (A0=1) – по лініях D[8:15]. Обмін словами з парною адресою відбувається по усім мініям шини даних. При адресації слова з непарною адресою процесор автоматично виконує два послідовні шинні цикли: перший по лініях D[8:15], другий — по D[0:7]. Частина шини даних по якій передаються дані визначається сигналами A0 и BHE#.

Синхронізація процесора здійснюється зовнішнім сигналом CLK, частота якого ділиться надвоє внутрішнім подільником для отримання сигналу внутрішньої синхронізації PCLK (Processor Clock). На зовнішні виводи процесора сигнал PCLK не виводиться, але його виробляє тактовий генератор i82284, синхронізуючись з сигналами S0# і SI#. Максимальна пропускна спроможність локальної шини процесора – одне слово за кожні два цикли внутрішньої частоти PCLK.

У процесорі можливе застосування конвеєрної адресації (Pipelined Addressing), вона передбачає можливість початку нового циклу (фази іден-

тифікації і адресації), до завершення обміну даними попереднього циклу. Конвеєрна адресація (рис. 2) дозволяє скоротити кількість тактів очікування через швидший початок виконання дешифрації адреси пам'яттю або портом вводу-виводу. Конвеєрний режим дозволяє контролерам пам'яті ефективно використовувати посторінковий режим звертання до динамічної пам'яті та чергування її банків. Процесор i80286 не фіксує код адреси на шині протягом поточного циклу обміну. Часова діаграма читання з пам'яті, або портів вводу-виводу зображена на рис. 2. Часова діаграма підтвердження переривання подібна часовій діаграмі на рис. 2, проте при підтвердженні переривання два рази здійснюється читання з видачею коду на лінії COD/INTA#, M/IO#, S1#, S0#, котрий відповідає сигналу INTA, адресна шина приймає довільний стан, а дані зчитані у першому циклі ігноруються. Часова діаграма запису у пам'ять та порти вводу-виводу зображена на рис. 3.

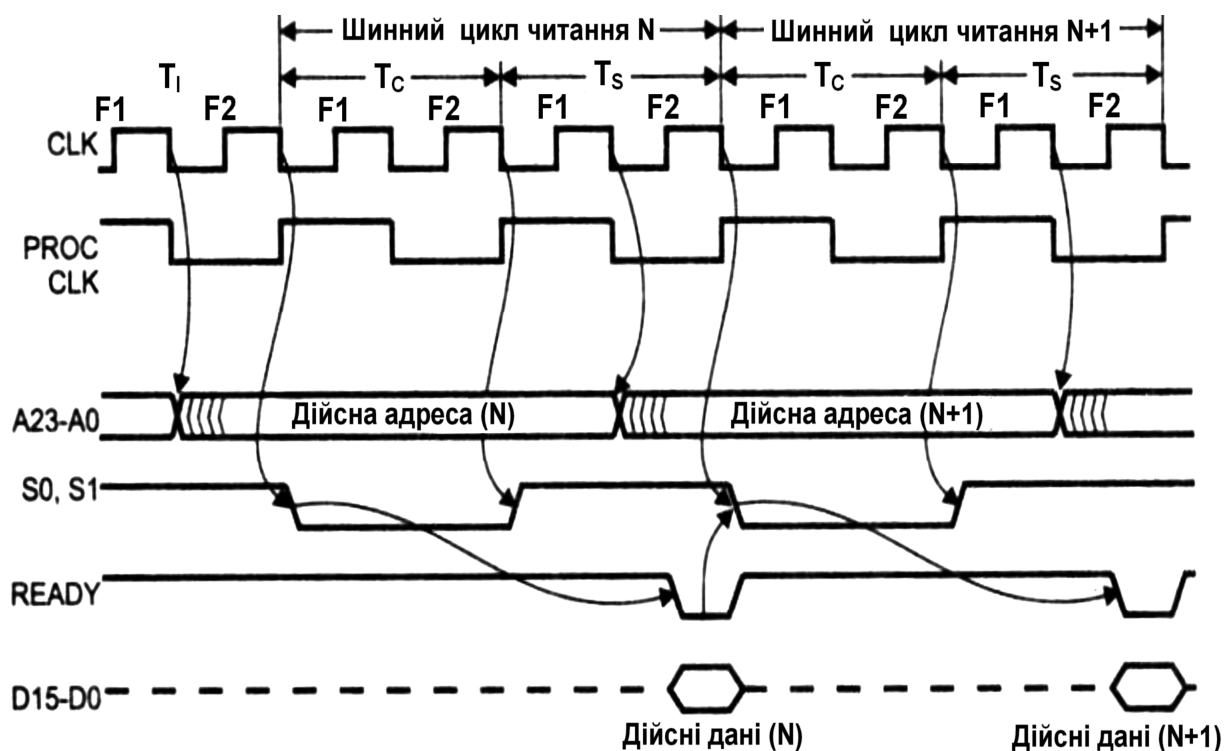


Рис. 2. Конвеєрні цикли читання МП i80286

Формування керуючих сигналів системної шини IOR#, IOW#, MEMR#, MEMWR#, INTA# і ALE виконує зовнішня мікросхема i82288 або її функціональний аналог.

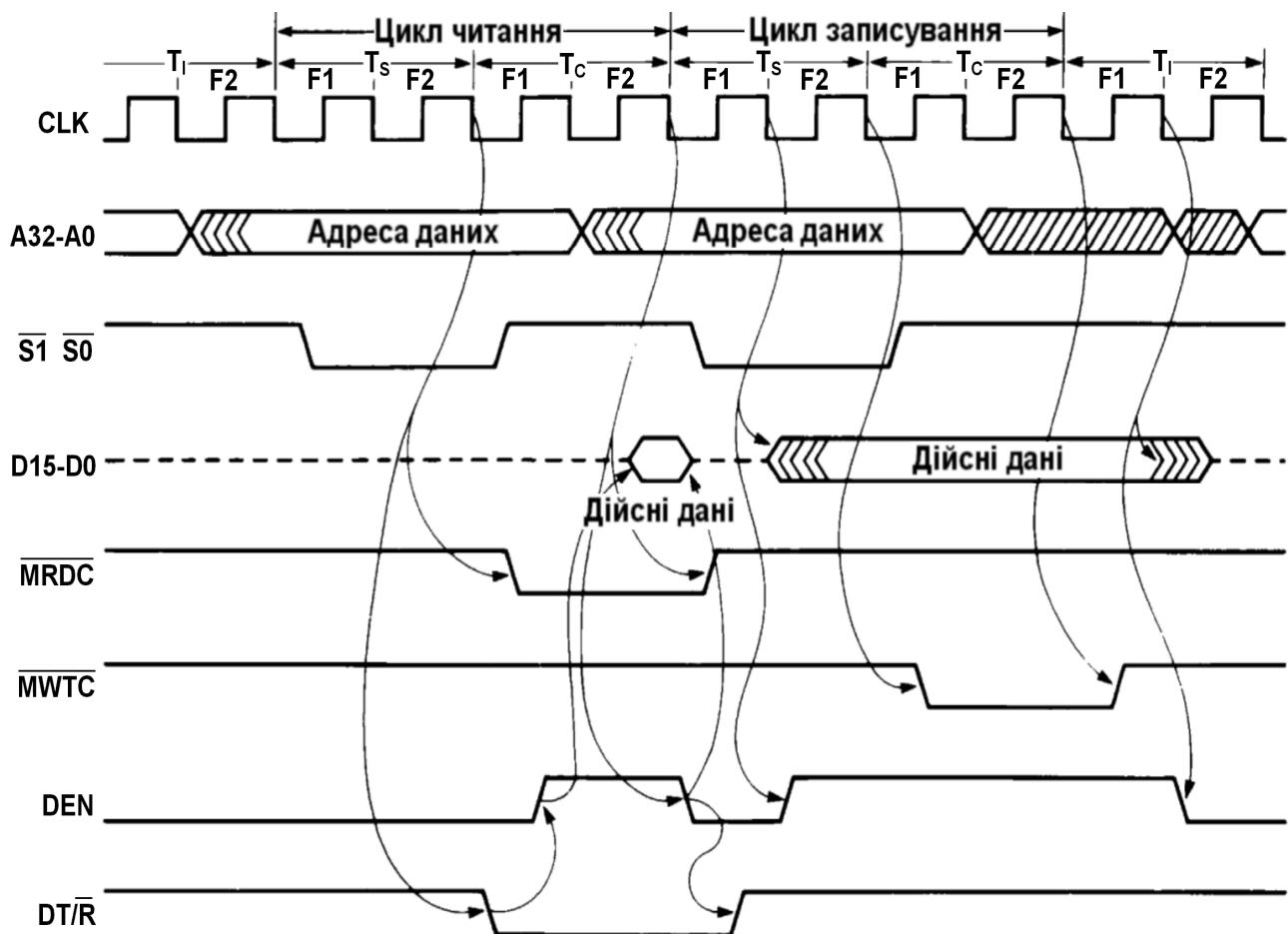


Рис. 3. Конвеєрні цикли читання та запису МП i80286

Керування часовою діаграмою за допомогою сигналу $READY\#$ дає змогу затримувати (Command delay) і розтягувати (Command extension) цикли обміну. Розтягування за необхідності подовжує цикл обміну для відносно повільних зовнішніх пристроїв вводу-виводу і пам'яті. Зовнішні схеми за допомогою затримування видачі сигналу $READY\#$ можуть розтягувати цикл на будь-який час. Формування сигналу $READY\#$ звичайно здійснюється мікросхемою i82288 (або її аналогом).

Необхідність затримування і розтягування циклів з'явилась у зв'язку з суттєвим підвищенням продуктивності локальної шини процесора відносно i8086. Процесор i80286 здатен генерувати шинні цикли з мінімальним періодом у два такти внутрішньої частоти. При частоті процесора 16 МГц, шинні цикли тривають 250 нс. Сповна цією продуктивністю може скористатися лише пам'ять і порти вводу-виводу, під'єднанні до локальної шини. Пристрої,

під'єднанні до шини ISA, такого темпу не витримують. Крім того, вони як правило вимагають визначеного часу відновлення (Recovery Time), пасивного стану керуючих сигналів шини між адресованими їм командами. Керування затримуванням та розтягуванням циклів, залежно від типу і адреси звернення, покладене на мікросхеми системної плати. Вибір реального чи захищеного режиму роботи МП ніяк не визначається на часових діаграмах.

2. Регістри МП i80286

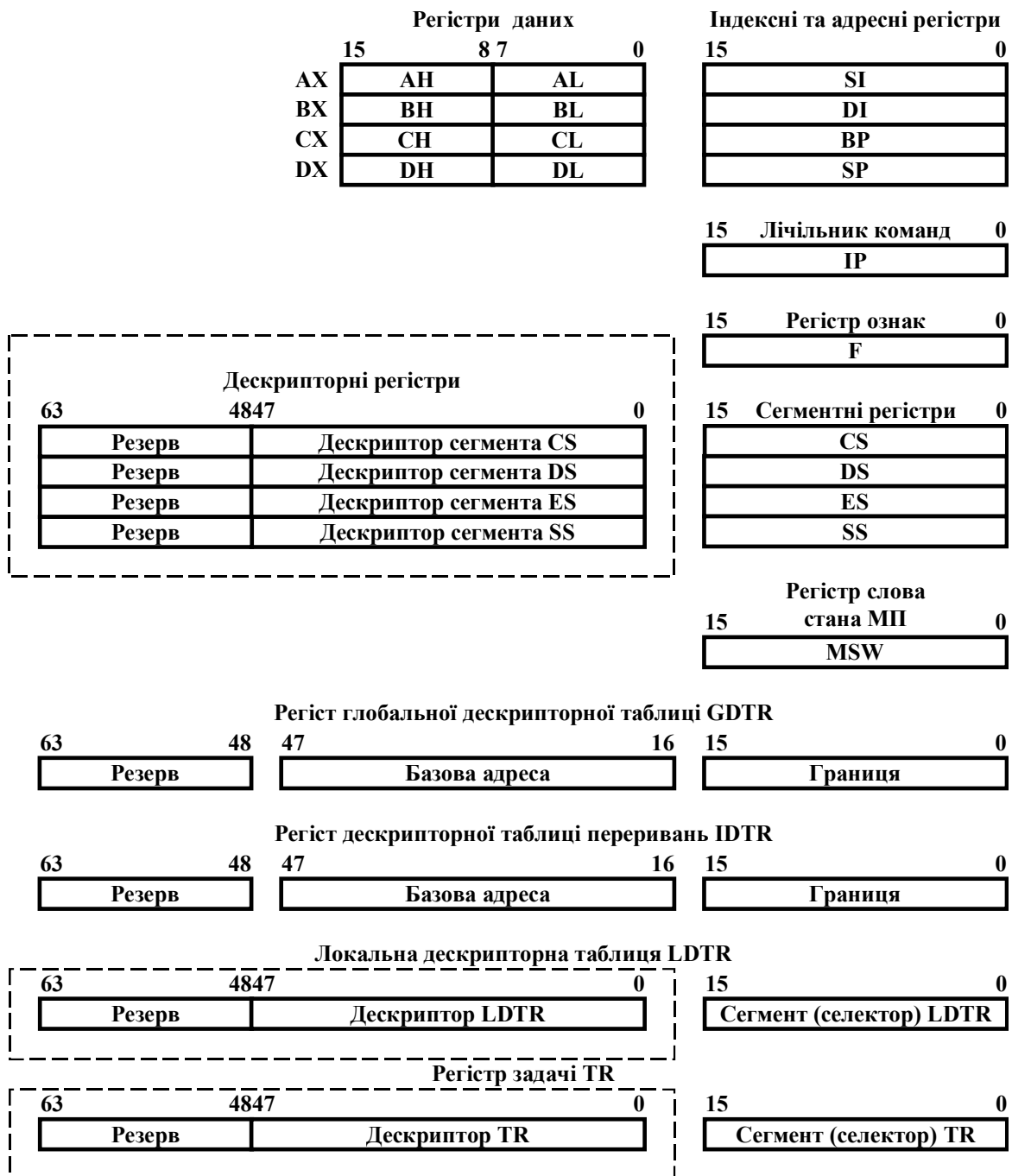


Рис. 4. Регістри МП i80286

У процесорі i80286 є 19 програмно доступних регістрів, з них 14 мають таку саму назву як і в i8086, а решта полегшує використання багатозадачних операційних систем (рис. 4). Вміст цих регістрів можна прочитати, або змінити певними командами. Зауважимо, що мікропроцесор містить також і "невидимі регістри", які зберігають різну інформацію для роботи процесора та пришвидшують його роботу.

Регістри можна об'єднати в групи за схожістю виконуваних ними функцій. У першу групу, названу групою регістрів загального призначення, входять регістри AX, BX, CX, DX. Вони призначені в основному для збереження даних. Регістри BX і DX можна додатково використовувати як адресні: регістр BX – як адреса зміщення байта чи слова в оперативній пам'яті, регістр DX – як адреса порту вводу-виводу. Кожен з цих регістрів має свої особливості. Наприклад, регістр AX завжди є одним з операндів команди множення, регістр CX використовується у якості лічильника у команді LOOP, DX – як розширення регістра AX у командах множення і ділення. Ці регістри є 16-розрядними, і складаються з двох одnobайтових регістрів кожен: AX – з AH і AL, BX – з BH і BL і т.д.

Наступну групу утворюють шістнадцятирозрядні регістри SP, BP, SI, DI. Їх називають групою адресних й індексних регістрів. Вони можуть використовуватися в якості адресних або індексних регістрів при обробці масивів даних, а також у якості шістнадцятирозрядних регістрів загального призначення (крім SP).

Шістнадцятирозрядні регістри CS, DS, SS, ES утворюють групу сегментних регістрів. У мікропроцесорі i80286 доступ до даних і коду програми в пам'яті здійснюється через сегменти розміром максимум 64 кбайт. Регістр CS визначає початок сегмента кода; регістр DS – початок сегмента даних; регістр SS – початок сегмента стека, регістр ES – початок сегмента додаткових даних.

У мікропроцесорі i80286 з'явилася можливість розміщувати таблицю векторів переривань у довільному місці оперативної пам'яті, а не обов'язково на самому початку, як у процесорі i8086. Для цього є спеціальний 40-бітний регістр IDTR. Він визначає початок і розмір таблиці векторів переривань. i80286 володіє

також спеціальними командами його читання і запису.

Шістнадцятирозрядний регістр IP служить для збереження адреси зміщення наступної команди, що виконується, в межах сегмента коду, а регістр F (регістр ознак) – для збереження ознак режиму роботи та ознак результатів останніх арифметичних чи логічних операцій.

У мікропроцесорі i80286 з'явився новий шістнадцятирозрядний регістр MSW, названий регістром слова стана, чи регістром стана. Він призначений для індикації режиму роботи МП та його зміни. Перехід у захищений режим роботи здійснюється встановленням біта PE в 1 у цьому регістрі. Формат регістра MSW зображений на рис. 5 Лише 4 біти MSW впливають на роботу МП, їх призначення наведено у таблиці 3.

15							8		7		0				
1	1	1	1	1	1	1	1	1	1	1	1	TS	EM	MP	PE

Рис. 5. Формат регістра MSW

Таблиця 3. Призначення бітів MSW

Біт MSW	Призначення
PE	Дозвіл захисту. Одиничне значення біта вмикає захищений режим роботи
MP	Моніторинг співпроцесора. Встановлення біта в 1 дозволяє викликати виключення #NM "співпроцесор не існує" при першій ж інструкції співпроцесора після переключення задач
EM	Емуляція співпроцесора. Встановлення біта в 1 дозволяє викликати виключення #NM при всіх інструкціях співпроцесора. Співпроцесор емулюється програмно
TS	Перемикання задач. Встановлення біта в одиницю указує, що відбувалось перемикання задач. Біт використовується для визначення стану співпроцесора

Регістр ознак F призначений для збереження ознак статусу та керування мікропроцесором. Формат регістра ознак зображено на рис. 6.

15							8	7						0
0	NT	IOPL	OF	DF	IF	TF	SF	ZF	0	AF	0	PF	1	CF

Рис. 6. Формат регістра ознак

Усі ознаки розділяються на ознаки статусу та ознаки керування. Ознаки

статусу визначають властивості результату виконання останньої арифметичної команди, команди маніпуляції бітами, або команди порівняння. Команди умовного переходу (та деякі інші), перевіряють ознаки статусу і дають різні результати залежно від їх стану. Ознаки керування призначені для зміни виконання деяких команд, чи режимів роботи МП. Наприклад, команда перегляду рядка може мати напрямок до більших чи менших адрес залежно від стану ознаки напрямку. Призначення бітів регістра ознак наведено у таблиці 4.

Таблиця 4. Призначення бітів регістра ознак

Назва біта	Призначення
NT	Ознака вкладеної задач. "0" - поточна задача є вкладеною, "1" – поточна задача не є вкладеною
IOPL	Рівень привілеїв вводу-виводу. Два біти визначають один з 4-х рівнів. Стан "00"(нульовий рівень) відповідає найвищому стану пріоритету, стану "11" (третій рівень) – найнижчому
OF	Ознака переповнення. Біт встановлюється, коли результат арифметичної операції не поміщається у заданій комірці
DF	Ознака напрямку операцій в командах обробки стрічок та послідовностей. Нуль указує на обробку з автоінкрементом (наступний операнд розміщений у комірках з більшими адресами), одиниця на обробку з автодекрементом (наступний операнд розміщений у комірках з меншими адресами)
IF	Дозвіл переривання. Одиничне значення біта дозволяє обробку запиту переривання по лінії INT. Нульове значення забороняє її
TF	Ознака трасування. Коли біт установлений у стан 1 після виконання кожної команди відбувається переривання типу INT 1
SF	Ознака знака. Одиничне значення біта указує на від'ємний результат останньої арифметичної або логічної операції. Фактично при встановленні знака МП вважає усі числа знаковими та поданими у додатковому коді. Знак визначається старшим бітом результату
ZF	Ознака нуля. Одиничне значення біта указує на нульовий результат останньої арифметичної або логічної операції
AF	Ознака додаткового перенесення. Одиничне значення біта указує на перенесення із третього у четвертий біт при виконанні останньої арифметичної операції. Ознака використовується при виконанні
PF	Ознака парності. Одиничне значення указує, що у молодшому байті результату парне число одиниць, нульове – непарне
CF	Ознака перенесення. Одиничне значення указує на наявність перенесення із старшого біта результату

Для захисту сегментів коду та даних в захищеному режимі роботи МП використовує спеціальні структури, що носять назву дескрипторних таблиць. Існує дві дескрипторні таблиці: локальна та глобальна. Визначення положення і розміру глобальної дескрипторної таблиці здійснюється 40-розрядним регістром GDTR, для визначення положення локальної дескрипторної таблиці використовується шістнадцятибітний регістр LDTR.

Шістнадцятирозрядний регістр TR служить для організації багатозадачної роботи МП у захищеному режимі. Цей регістр служить селектором сегмента стана задачі.

3. Переривання у МП i80286

У МП i80286 є декілька типів переривань: програмні переривання, апаратні зовнішні переривання, апаратні внутрішні переривання (виключення). Програмні переривання, це швидкий метод виклику підпрограм операційної системи. Програмні переривання поділяють на переривання за відлагоджувальними точками, за переповненням та за вектором.

Переривання за відлагоджувальною точкою генеруються за однобайтовою командою INT3. Основна задача цієї команди дати можливість зупинити програму, що відлагоджується у будь-якому місці програми.

Переривання за переповненням генерується за командою INTO. Якщо ознака переповнення OF рівна 1 у момент виконання команди, то за командою INTO здійснюється перехід до обробки переривання за вектором (номером) 4.

Переривання за вектором дозволяє викликати будь-яке переривання. Якщо у процесі виконання програми у ній зустрінеться команда INT x, де x - число від 0 до FFh, то відбудеться переривання з номером x.

Апаратні зовнішні переривання виникають при надходженні сигналу на вхід запиту переривання. У МП i80286 є два входи запитів переривань: запит маскованих переривань INT та запит немаскованих переривань NMI. Масковані переривання можуть бути заблоковані скиданням біта дозволу переривань IF у регістрі ознак МП. При надходженні на вхід INT логічної

одиниці, якщо переривання дозволені, здійснюється процедура підтвердження переривання. У процесі виконання підтвердження переривання пристрій, котрий запросив переривання, передає МП вектор переривання. На відміну від маскованих немасковані переривання виникають завжди при переході лінії NMI у стан логічної 1. Запити переривань сприймаються та обробляються після закінчення виконання команди.

До внутрішніх апаратних переривань відносять переривання по помилці ділення та переривання покрокової роботи. Переривання по помилці ділення відбувається після виконання команди ділення (DIV або IDIV), якщо результат ділення перевищує по модулю максимальну допустиму величину. Покрокове переривання призначено для відлагодження програми, воно відбувається після виконання кожної команди програми, якщо біт TF регістра ознак рівний 1. Усі згадані переривання можуть бути викликані у реальному та захищеному режимах роботи.

У процесорі i80286 з'явився новий клас переривань, що носять назву "виключення". На відміну від "класичних" внутрішніх апаратних переривань, виключення можуть бути оброблені до закінчення виконання команди, що спричинила появу виключення.

Усі виключення розділяють на три класи:

- ◇ Відмова (fault) – виключення, що виявляється та обслуговується до закінчення виконання команди, котра викликала виключення. Після виконання підпрограми обслуговування відмови можливе повторне виконання команди.
- ◇ Пастка (trap) – виключення, що виявляється та обслуговується після команди, що його викликала.
- ◇ Аварія (abort) – виключення, що не дозволяє точно встановити команду, котра його викликала. Після завершення обслуговування такого виключення подальше виконання програми неможливе.

Множина виключень залежить від режиму роботи МП. Інформація про усі виключення та переривання зображені у таблиці 5. У таблиці використо-

вуються позначення: F – виключення типу "відмова"; A – виключення типу "аварія"; T – виключення типу "пастка "; AI – зовнішнє апаратне переривання; PI – програмне переривання; EC – перед викликом підпрограми обробки виключення у стек заноситься ідентифікаційний код виключення, EC = 0 - перед викликом підпрограми обробки виключення у стек заноситься код 0000h.

Таблиця 5. Переривання та виключення

Функція	Номер вектора	Тип
Переривання та виключення, наявні у реальному та захищеному режимах		
Переривання за помилкою ділення (#DE)	0	F
Покрокове переривання (#DB)	1	T
Немасковане переривання	2	AI
Переривання за відлагоджувальною точкою (#BP)	3	T
Переривання за переповненням (INTO) (#OF)	4	T
Виключення за контролем діапазону BOUND (#BR)	5	F
Недопустимий код операції (#UD)	6	F
Співпроцесор недоступний виконуваний програмі (#NM)	7	F
Апаратні масковані переривання. Номер вектора X передається у другому циклі підтвердження переривання.	X	AI
Програмні переривання за командою INT X	X	PI
Співпроцесор звертається до даних за межами сегмента	9	F
Переривання та виключення, наявні лише у реальному режимі		
Вектор відсутній у таблиці векторів переривань	8	A, EC=0
Сегмент стека переповнений (#SS)	0Ch	F, EC
Процесор звертається до даних або коду за межами сегмента (#GP)	0Dh	F, EC
Переривання та виключення, наявні лише у захищеному режимі		
Подвійне виключення (#DF)	8	A, EC=0
Недійсний сегмент стану задачі (#TS)	Ah	F, EC
Сегмент кода, або даних відсутній у пам'яті (#NP)	0Bh	F, EC
Сегмент стека переповнений, або відсутній (#SS)	0Ch	F, EC
Загальна відмова захисту (#GP)	0Dh	F, EC

Фірма Intel зарезервувала переривання з номерами 0 - 20h (0 - 31) для подальшого використання у старших моделях процесорів, проте у реальному режимі роботи, більшість цих векторів використовується операційними системами реального режиму роботи.

У випадку виникнення переривання або виключення у залежності від його типу формується номер. Номери векторів переривань наведені у таблиці 5. По номеру вектора здійснюється пошук адреси підпрограми обробки переривання у таблиці векторів переривань IDT.

Формат IDT близький до формату таблиці векторів переривань МП i8086. У таблиці IDT у реальному режимі роботи по послідовних адресах розміщені зміщення та сегмент відповідної підпрограми обробки переривання. У захищеному режимі роботи на позиції сегмента розміщений селектор, котрий посилається на шлюзи задач (Task Gate), шлюзи переривань (Interrupt Gate), або шлюзи пасток (Trap Gate).

Таблиця IDT може розміщатись будь-де у пам'яті. Для встановлення початкової адреси таблиці векторів переривань МП використовує 40-розрядний регістр IDTR, формат та призначення розрядів котрого схожі на формат регістра GDTR.

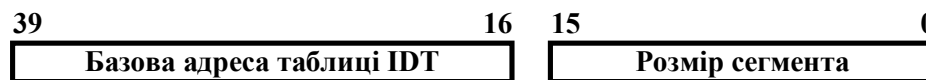


Рис. 7. Формат регістра IDTR

4. Реальний режим роботи

Реальний режим роботи призначений для забезпечення найбільшої сумісності з МП i8086. Адресація пам'яті у реальному режимі роботи здійснюється аналогічно до i8086, ніякого захисту програм та даних немає.

Для адресації пам'яті МП використовує логічні та фізичні адреси. Логічна адреса є 32-бітною незалежно від режиму роботи МП. Вона містить 16-розрядний сегмент та 16-розрядне зміщення. Логічну адресу звичайно записують у формі: сегмент:зміщення. У реальному режимі мікропроцесор звертається до пам'яті, генеруючи 20-розрядні (точніше, "майже 20-розрядні") фізичні адреси. Формування фізичної адреси здійснюється по правилу зображеному на рис. 8: сегментна частина зсувається на 4 розряди вліво та додається до зміщення. Отже адреси початку сегментів завжди починаються з числа, кратного 16, а кожен сегмент пам'яті має розмір 64 кбайти. Легко

довести, що одній фізичній адресі відповідає 16 384 різних логічних адрес. Наприклад: фізичній адресі 00400h (0000 0000 0100 0000 0000b) відповідають логічні адреси 0040:0000 та 0000:0400.

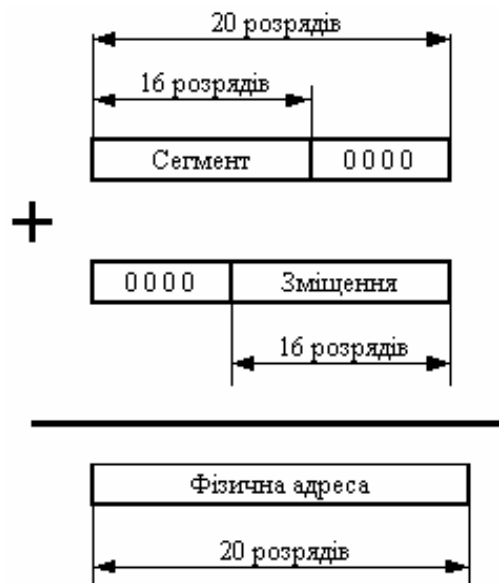


Рис. 8. Формування фізичної адреси у реальному режимі роботи

Хоча адресація пам'яті в реальному режимі роботи здійснюється по алгоритму використаному у процесорі i8086, але розробники МП лишили можливість адресувати дані за межами першого мегабайта. Адреса 0F0000:0FFFF відповідає максимально можливій фізичній адресі 0FFFFFFh. Але, використовуючи 16-розрядні регістри процесора, є можливість задати і більші значення логічної адреси, наприклад, 0FFFFFFh:0010h. У даному МП при використанні указаній логічної адреси відбудеться звертання по фізичній адресі 100000h, котра знаходиться за межами першого мегабайта (у МП i8086 було б звертання по адресі 00000h). Тобто у реальному режимі з'являється ще один додатковий сегмент пам'яті, що лежить вище межі першого мегабайта. Цей сегмент називається областю верхньої пам'яті (High Memory Area). Йому відповідає діапазон логічних адрес від 0FFFFFFh:0010h до 0FFFFFFh:0FFFFFFh. Розмір області складає 64 кбайта без 16 байт.

Для досягнення повної сумісності з МП i8086 на системній платі є логічний елемент, що може примусово обнулити лінію A20, проте можливість отримання додаткової пам'яті звичайно переважає можливість звертання до

початкових адрес МП і8086 дуже нестандартним чином і таким блокуванням не користуються. Існує також можливість завантаження у сховані регістри МП базової адреси, що знаходиться за межами першого мегабайта по відлагоджувальній команді LOADALL, котра завантажує абсолютно усі регістри МП даними з структури розміщеної по адресам 80:0h-80:66h.

У реальному режимі роботи будь-яка програма може встановити довільні значення в сегментні регістри і адресувати будь-яку ділянку пам'яті. Зокрема, будь-яка програма може навмисно чи через помилку зруйнувати області даних, що належать операційній системі.

У реальному режимі передавання керування виконується за допомогою команд JMP, CALL, INT, RET, IRET, а також при виникненні переривань. При внутрішньосегментному передаванні керування у регістр IP заноситься нове значення, а регістр CS не модифікується. Міжсегментне передавання і одночасно змінює регістри CS і IP, а також у деяких випадках і регістр ознак F (переривання і команди RET, IRET).

Внутрішньосегментне передавання керування виконується командами JMP, CALL, RET, а міжсегментна передача керування – командами JMP, CALL, INT, RET, IRET і у випадку виникнення переривань.

У реальному режимі будь-якій програмі дозволяється робити перехід на будь-яку адресу, або викликати будь-які підпрограми та переривання. Є можливість здійснити перехід на команди початкової ініціалізації системи або спробувати виконати дані. Єдина умова для успішного виклику підпрограми – знання адреси підпрограми (сегмента і зміщення) і формату переданих даних.

Отже два основні недоліки реального режиму:

- ◇ обмежений адресний простір;
- ◇ вільний доступ для будь-яких програм, до будь-яких областей даних, що становить потенційну небезпеку для цілісності операційної системи.

Цих недоліків повністю позбавлена схема адресації пам'яті, яка використана у захищеному режимі.

5. Захищений режим роботи

Захищений режим передбачає розширений адресний простір фізичної і віртуальної пам'яті, механізми захисту пам'яті, нові операції з підтримки операційних систем і віртуальної пам'яті. Захищений режим забезпечує віртуальний адресний простір 1 Гбайт для кожної задачі, котрий відображається на фізичний адресний простір у 16 Мбайт. Віртуальний простір може бути більший фізичного, тому що будь-яке використання адреси, що не розподілена у фізичній пам'яті, створює виняткову ситуацію, котра обробляється операційною системою.

Переключення МП у захищений режим відбувається за командою LMSW, за якою біт PE регістра MSW встановлюється в 1.

У цьому режимі вміст сегментного регістра призначений для пошуку базової адреси у спеціальній таблиці базових адрес, що розміщена у пам'яті. Таку таблицю називають дескрипторною, її рядок складається з 48 бітів (ще 16 старших біт зарезервовані), а сам сегментний регістр у захищеному режимі називають селектором. Формат селектора зображено на рис. 9.

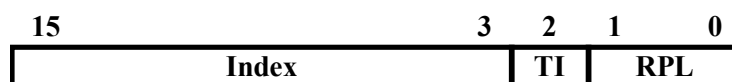


Рис. 9. Формат селектора

Два молодші розряди селектора (RPL), визначають чотири рівні привілеїв, що запитує програма при доступі до сегмента. Привілеї дозволяють використовувати одні й ті ж самі сегменти для різних програм, або у різних місцях програми по різному. Наприклад, ядро операційної системи, що має найвищий рівень привілеїв може змінювати дані про всі активні прикладні програми та використовувати ними ресурси, а прикладна програма, котра працює з найнижчим привілеєм, може лише зчитувати дані про саму себе.

Наступний біт селектора (TI) указує тип дескрипторної таблиці. Мікропроцесор i80286 використовує дві дескрипторні таблиці:

- ◇ глобальну дескрипторну таблицю GDT, яка адресує сегменти, що можуть бути доступні будь-якій задачі;

- ◇ локальну дескрипторну таблицю LDT, що адресує сегменти, доступні лише конкретній задачі.

Якщо біт TI рівний 0 вибирається глобальна дескрипторна таблиця, якщо 1 – локальна дескрипторна таблиця. Старші тринадцять розрядів селектора (Index), вказують на номер 48-розрядного запису з інформацією про базову адресу, розмір сегмента та права доступу. Формат дескрипторної таблиці та самих дескрипторів розглянемо нижче.

У захищеному режимі фізична адреса формується за такою схемою:

- ◇ МП за вмістом селектора, що міститься в одному із сегментних регістрів, адресує запис у дескрипторній таблиці;
- ◇ з дескрипторної таблиці зчитується значення дескриптора, що описує властивості сегмента та містить 24-розрядну базову адресу;
- ◇ перевіряється можливість виконання операції з сегментом та межі сегмента;
- ◇ базова адреса сумується із зміщенням, внаслідок чого формується фізична адреса байта або слова в пам'яті (рис. 10).

Щоб уникнути звертання до пам'яті, де розміщена дескрипторна таблиця, при кожному звертанні до даних, елементи дескрипторної таблиці, які відповідають вмісту селекторів, запам'ятовуються у тіньових дескрипторних регістрах МП. Заповнення тіньового дескрипторного регістра МП відбувається при першому звертанні до пам'яті в використанні відповідного селектора.

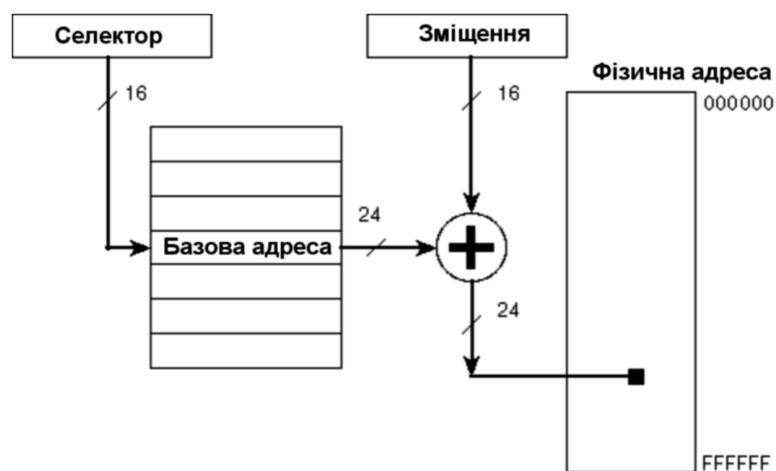


Рис. 10. Формування фізичної адреси у захищеному режимі роботи

5.1. Регістри дескрипторної таблиці

Розмір та розміщення глобальної дескрипторної таблиці визначається 40-розрядним регістром GDTR. Формат регістра зображено на рис. 11.

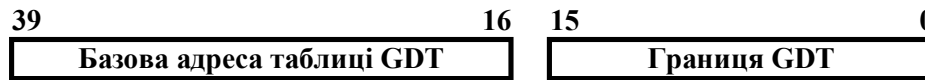


Рис. 11. Формат регістра GDTR

Розмір та розташування локальної дескрипторної таблиці визначає 16-розрядний регістр LDTR, в якому зберігається селектор для вибору сегмента пам'яті з таблицею LDT. Формат селектора LDT є загальним, (рис. 9) проте біт TI може бути рівний лише 0. Формат дескриптора LDT має вигляд, зображений на рис. 12.



Рис. 12. Формат дескриптора LDT

При адресації будь-якого сегмента, використовують глобальну або локальну дескрипторну таблиці. Кожний запис у таблиці носить назву дескриптор і складається із 64 біт з котрих використовуються лише 48.

5.2. Дескриптори

Дескриптори призначені збереження інформації про сегменти даних, коду та сегменти службового призначення. Дескриптори сегментів коду, даних та деяких службових сегментів (дескриптори шлюзів та задач) поміщають у дескрипторну таблицю.

У дескрипторі задається рівень привілеїв, що може отримати програма при доступі до сегменту. Крім того у дескрипторі задається тип сегмента, його доступність, а також властивості сегмента. Наявність у дескрипторах сегментів прав доступу забезпечує захист від запису у сегменти коду, заборону виконання сегменту даних, контроль за переповненням сегментів стеку, тощо.

Усі дескриптори можна розділити на дескриптори даних, дескриптори коду та службові дескриптори. Незалежно від призначення дескриптори мають формат зображений на рис. 13.

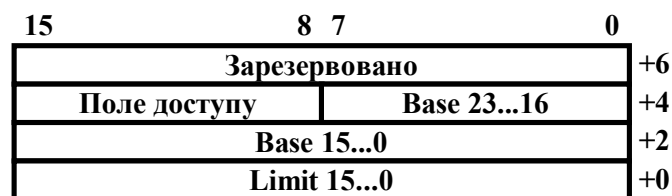


Рис. 13. Формат дескриптора

Тип дескриптора визначається бітами у байті, котрий носить назву "Поле доступу". Межа сегмента визначається полем Limit 15..0, вона визначає максимальне або мінімальне зміщення котре може бути використане при адресації даних. Фізична адреса початку сегмента визначається полем Base (Base23..16, Base15..0). Старші два байти у дескрипторі процесора i80286 були зарезервовані для подальшого використання. Для сумісності з іншими МП у ці розряди повинні бути записані нулі.

На рис. 14 наведені формати поля доступу всіх типів дескрипторів: дескрипторів сегментів коду, сегментів даних і системних сегментів. У всіх дескрипторів три старших біти поля доступу мають однакове призначення.

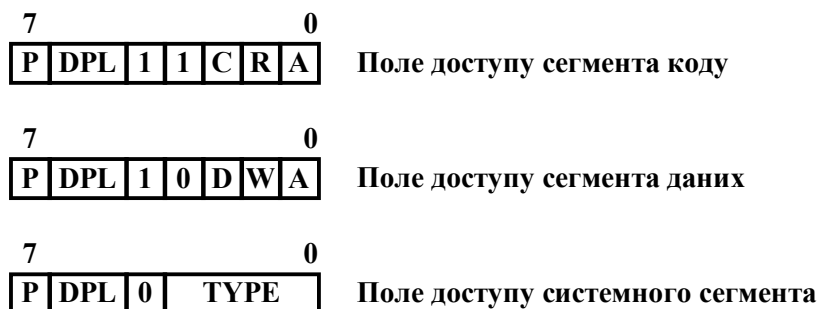


Рис. 14. Формати поля доступу дескриптора

Біт Р призначений для організації віртуальної пам'яті. Біт Р указує на присутність сегмента в пам'яті. Для тих сегментів, що знаходяться у пам'яті, цей біт повинен бути встановлений в 1. Будь-яка спроба програми звернутися до сегмента пам'яті, у дескрипторі якого біт Р встановлений в 0, призведе до виключення (11). У процесі обробки виключення можливе або ігнорування звертання до сегмента, або зчитування даних з жорсткого диска і заповнення сегмента необхідними даними, або емуляція виконання команди, котра зчитувала дані цього сегмента, або аварійна зупинка програми.

Поле DPL займає два біти та задає атрибути привілей сегменту. Воно

контролює можливість використання сегмента задачею.

Четвертий біт визначає властивість сегмента. Якщо сегмент є сегментом даних, або коду біт рівний 1, якщо ж сегмент є системним - біт рівний 0.

Загальним для усіх типів сегментів є також біт А (молодший біт поля доступу), котрий називають бітом доступності сегмента. Для всіх сегментів, до котрих було звертання, або виконувалась команда тестування селектора біт автоматично встановлюється в 1. У випадку, коли звертань до сегмента ще не було біт встановлюється в 0.

Третій біт поля доступу сегментів коду та даних дозволяє розрізнити сегменти даних та коду. У сегментів коду цей біт рівний 1, у даних – 0.

Біт С поля доступу сегментів коду визначає підпорядкованість сегментів. Якщо $C = 0$, то код з цього сегмента може виконуватись лише коли $DPL = CLP$. Такі сегменти називаються непідпорядкованими. У випадку коли $C = 1$, код з цього сегмента може виконуватись для всіх задач, якщо виконується умова $CPL \geq DPL$. У такому випадку код буде виконуватись з рівнем привілеїв, що задавався бітами CPL, а не DPL.

Поле доступу дескриптора сегментів коду містить також бітове поле R, яке називають бітом дозволу читання сегмента. Якщо цей біт встановлений в 1, програма може зчитувати вміст сегмента коду. У протилежному випадку процесор може тільки виконувати цей код.

Дескриптори сегмента даних мають бітові поля W і D. Поле W називають бітом дозволу записування даних в сегмент. Якщо цей біт встановлений в 1, поряд з читанням можливе і записування даних у даний сегмент. В іншому випадку при спробі читання виконання програми буде перервано.

Поле D задає напрямок розширення сегмента. Звичайний сегмент даних розширюється в область старших адрес (розширення вгору). Якщо ж у сегменті розміщений стек, розширення звичайно відбувається у зворотному напрямку – в область молодших адрес (розширення вниз). Для сегментів, у яких організуються стек, необхідно встановлювати поле D в 1.

Дескриптори системних сегментів у чотирьох молодих бітах містять поле

TYPE, що визначає тип системного сегмента. У таблиці 5 наведені можливі для цього поля значення.

Таблиця 6. Типи системних сегментів

Поле TYPE	Тип сегмента
0000	Заборонене значення
0001	Дескриптор TSS доступної задачі
0010	Сегмент LDT
0011	Дескриптор TSS зайнятої задачі
0100	Вентиль виклику
0101	Вентиль задачі
0110	Вентиль переривання
0111	Вентиль пастки (trap)
1000	Заборонене значення
1001	Зарезервовано
1010	
1011	
1100	
1101	
1110	
1111	

Дескриптори службових сегментів, наприклад, які описують розташування таблиці LDT чи сегмента стану задачі, не можна використовувати для читання чи запису, навіть якщо програма виконується у нульовому пріоритетному кільці. У випадку необхідності можна створити додаткові дескриптори, у яких ці ж сегменти описані як сегменти даних. Так само слід поступати, якщо необхідно виконати запис у сегмент коду.

5.3. Сегмент стану TSS та переключення задач

Для організації багатозадачної роботи мікропроцесора в захищеному режимі використовується ще один 16-розрядний регістр – регістр задачі TR, а в пам'яті наявна структура, котра носить назву "сегмент стану задачі" (TSS).

Формат регістра TR, що указує на сегмент стану задачі зображений на рис. 15. На рисунку введені позначення: Base - базова адреса сегменту TSS; Limit - границя TSS. Поле доступу має формат, що використовується для системних сегментів. Границя має бути такою, щоб у сегмент TSS поміщались

по крайній мірі усі стандартні поля. У випадку необхідності сегмент TSS може бути розширений, за допомогою зміни поля Limit. Формат даних, що знаходяться за адресами з зміщенням рівним або більшим 2CH визначається операційною системою.

15	8	7	0	
Зарезервовано				+6
Поле доступу		Base 23...16		+4
Base 15...0				+2
Limit 15...0				+0

Рис. 15. Формат дескриптора TSS

	2CH	
Селектор LDT	2BH	Стан процесора
Селектор DS	28H	
Селектор SS	26H	
Селектор CS	24H	
Селектор ES	22H	
DI	20H	
SI	1EH	
BP	1CH	
SP	1BH	
BX	18H	
DX	16H	
CX	14H	
AX	12H	
F	10H	
IP	0EH	Початковий стек для задач відповідного рівня
SS для CPL 2	0Ch	
SP для CPL 2	0BH	
SS для CPL 1	08H	
SP для CPL 1	06H	
SS для CPL 0	04H	
SP для CPL 0	02H	
TSS попередньої	00H	

Рис. 16. Сегмент стану задачі TSS

Сегмент стану задачі має формат зображений на рис. 16. Він зберігає копії всіх регістрів, які використовує мікропроцесор для тимчасового зберігання даних, формування адреси і тощо. Сегмент використовується в багатозадачних операційних системах для збереження стану МП у процесі

перемикання задач. Для кожної задачі операційна система створює свій сегмент стану. У процесі перемикання задач стан усіх регістрів МП зберігається у відповідному сегменті TSS, а потім відновлюється стан іншої задачі.

5.4. Захист у процесорі i80286

Процесор i80286 використовує гнучку і надійну схему захисту операційної системи і програм однієї від другої. У цій схемі використовують привілеї чотирьох рівнів - від 0 до 3. Найбільші привілеї відповідають рівню 0, зазвичай такими привілеями володіє ядро операційної системи. Мінімальні привілеї у користувальницьких програм (рівень 3). Рівні привілеїв часто називають кільцями захисту.

5.4.1. Рівні захисту

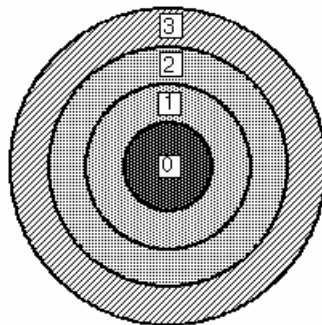


Рис. 17. Рівні захисту

Зазвичай використовують такий розподіл рівнів захисту:

- ◇ Рівень 0 – ядро операційної системи;
- ◇ Рівень 1 – програми обслуговування апаратури, драйвери, програми, що працюють з портами вводу-виводу;
- ◇ Рівень 2 – системи керування базами даних, програми операційної системи;
- ◇ Рівень 3 – прикладні програми, які запускає користувач.

Нескладні системи можуть використовувати не всі рівні, а тільки деякі чи навіть один. Наприклад, всі програми операційної системи можуть працювати з рівнем 0, а користувальницькі програми - з рівнем 3. Це варіант описаної вище схеми "супервізор-користувач". Найпростіші системи повністю реалізуються на нульовому рівні привілеїв, проте у таких системах захист відсутній.

Будь-який селектор має поле RPL (біти 0 і 1). У регістрі CS зберігається селектор поточного виконуваного сегмента коду. Цьому селектору відповідає дескриптор у таблиці GDT чи LDT. У дескрипторі, у байті доступу, розташовується поле DPL (біти 5 і 6). Зазначені поля беруть участь у механізмі захисту пам'яті.

Програма отримує від операційної системи поточний рівень привілеїв CPL, який може проаналізувати на підставі вмісту регістра CS. Коли операційна система готує програму для запуску, то формує у GDT чи LDT дескриптор, що описує сегмент коду програми. У цьому дескрипторі в полі DPL байта доступу проставляється номер рівня, на якому працюватиме дана програма – поточний рівень привілеїв CPL (Current Privilege Level). Тобто можливості програми визначаються умістом поля DPL у байті доступу.

Поточний рівень привілеїв копіюється у поле RPL селектора сегмента коду, завантаженого в регістрі CS. Програма завжди може проаналізувати свій поточний рівень привілеїв, виходячи зі значення поля RPL у регістрі CS. Однак, вона не зможе змінити свій рівень привілеїв простою заміною умісту поля RPL у сегменті CS.

Дескриптори, що описують сегменти даних, містять поле рівня привілеїв дескриптора DPL (Descriptor Privilege Level). Поле DPL містить привілеї, потрібні для доступу до сегмента даних. Перед тим, як звернутися до сегмента даних, програма повинна завантажити в один із сегментних регістрів селектор, що відповідає потрібному сегменту даних. У селекторі необхідно вказати поле рівня запитуваних привілеїв RPL (Requested Privilege Level).

Програмі буде наданий доступ до сегмента тільки в тому випадку, коли рівень привілеїв дескриптора запитуваного сегмента DPL не менше найбільшого зі значень поточного рівня привілеїв CPL і рівня запитуваних привілеїв RPL ($DPL \geq \max(CPL, RPL)$).

Якщо програма спробує отримати доступ до більш привілейованого, сегмента пам'яті, у момент виконання команди звертання буде викликане виключення "загальне порушення захисту". Єдине виключення становлять

підпорядковані сегменти коду (біт $C=1$ у полі доступу дескриптора), для котрих дозволено читання. Їх уміст може бути прочитаний задачею з будь-яким значенням CPL.

5.4.2. Межі сегментів

Програми реального режиму працюють завжди із сегментами розміром 64 кбайти. Якщо програма складається із кількох сегментів, то деякі з них можуть перекриватися. Існує потенційна небезпека, що у наслідок програмної помилки (наприклад, виходу індексу масиву за допустимі межі) відбудеться запис в інший сегмент.

У захищеному режимі процесор i80286 дозволяє створювати сегменти будь-якого розміру в межах 64 Кбайти. Крім того, він стежить за тим, щоб при адресації пам'яті не відбувався вихід за границі сегмента. Границю сегмента задає поле Limit в дескрипторі. Воно вказує зміщення останнього байта у сегменті. Інтерпретація поля Limit залежить від стану біта D поля доступу. Біт D рівний нулю вказує, що дані в сегменті розміщуються зі зміщенням від нуля до Limit включно, а D рівний одиниці – зі зміщенням від Limit до значення 0FFFFh. Звичайно $D = 0$ для сегментів даних.

Для сегментів стека необхідно встановлювати поле D у одиницю. Спроба запису за межами сегмента викликає виключення #GP "Загальне порушення захисту" або для сегментів стека #SS "Порушення межі сегмента стека". У процесі обробки виключення операційна система може, наприклад, виділити додаткову пам'ять для сегмента, або заборонити виконання команди.

5.4.3. Привілейовані команди

Для надійності роботи операційної системи необхідно обмежити використання звичайними програмами деяких команд процесора. Наприклад, звичайні програми не повинні мати доступу до команд завантаження системних регістрів GDTR і LDTR. У таблиці 7 наведений список привілейованих команд, що можуть виконуватися тільки в нульовому пріоритетному кільці, тобто тільки тими програмами, що мають найбільші привілеї ($CPL=0$). Якщо команда буде виконана на іншому рівні CLP буде викликане виключення #GP.

Таблиця 7. Привілейовані команди процесора i80286

Команда	Виконувані функції
LGDT	Завантаження регістра глобальної таблиці дескрипторів GDTR
LLDT	Завантаження регістра локальної таблиці дескрипторів LDTR
LIDT	Завантаження регістра таблиці дескрипторів переривань IDTR
LTR	Завантаження регістра задачі TR
LMSW	Завантаження слова стану машини MSW
CLTS	Скидання ознаки переключення задачі
LOADALL	Завантаження усіх регістрів МП
HLT	Зупинка процесора

У наслідок, наприклад, помилки програміста, у програмі може зустрітися код, що не відповідає жодній команді. У такому випадку процесор формує виключення #UD "Недійсний код команди".

Існують команди, які повинні виконуватись не тільки в нульовому кільці, але використання яких повинно бути заборонено або модифіковано для програм, котрі мають рівень привілеїв, нижчий заданого для усієї системи в цілому. Наприклад, необхідно обмежити звичайні програми у використанні команд вводу/виводу: IN, OUT, INS, OUTS. За допомогою цих команд програма може, наприклад, перепрограмувати контролери переривань і прямого доступу до пам'яті, що відкріє шлях до безпосередньої модифікації пам'яті за фізичними адресами. Також не слід дозволяти звичайним програмам використовувати команди, що змінюють ознаку дозволу переривань у регістрі ознак (F).

Регістр ознак F процесора i80286 у розрядах 12-13 містить поле IOPL (Input-Output Privilege Level), яке визначає найменш привілейований рівень, котрому якому дозволено використовувати команди вводу-виводу та роботи з бітом IF. Програми, що не працюють на нульовому рівні, не можуть самі модифікувати поле IOPL регістра F. При виконанні команд, що завантажують регістр ознак (IRET і POPF) не при нульовому рівні CPL, поле IOPL не модифікується. Аналогічно, без зміни, залишається ознака дозволу переривань IF.

Якщо програма використовує команди вводу-виводу і має $CPL > IOPL$ у

момент виклику команди роботи буде викликано виключення #GP "Загальне порушення захисту". У процесі його обробки операційна система може або сама провести обмін з портом вводу-виводу (можливо по іншій адресі, емулюючи неіснуюче, або зайняте іншою програмою обладнання), або заблокувати обмін, повернувши керування на адресу, що йде після команди роботи з портом, або просто зупинити програму.

Якщо програма використовує команди скидання та встановлення біта дозволу переривань (CLI та STI) і має $CPL > IOPL$ у момент виклику команди роботи буде викликано виключення #GP "Загальне порушення захисту". У процесі його обробки операційна система може або заборонити переривання, або заборонити надходження даних у програму, або повернути керування програмі без будь-яких змін IF.

5.5. Міжсегментні переходи

Для міжсегментного передавання керування використовуються команди JMP, CALL, INT, RET, IRET. У командах JMP та CALL вказується адреса, куди буде відбуватись перехід у форматі селектор:зміщення. У захищеному режимі передавання керування можливе як напряду на сегменти коду, так і через шлюзи.

Передача керування з використанням селекторів сегментів коду можливе лише на сегмент, котрий має рівень привілеїв DPL, як і CPL задачі (якщо у полі доступу сегмента з відповідним селектором біт C рівний 0), або на сегмент у котрого $CPL \leq DPL$ (якщо у полі доступу сегмента з відповідним селектором біт C рівний 1). У будь-якому випадку рівень привілеїв програми не зміниться. Передача керування на сегменти коду з рівнем DPL, котрий не підпадає під указані умови не відбувається.

Перехід до виконання коду високим рівнем привілеїв можливий з використанням шлюзів (Gate), або при переключенні задач. Для забезпечення переключення задачі необхідно, щоб селектор у команді переходу вказував на сегмент стану іншої задачі, або на шлюз задачі.

Формат селектора шлюзу співпадає з форматом селектора кодового сегмента. Він є поєднанням селектора котрий поміщається у команді переходу чи виклику у позицію сегмента та дескриптора, котрий визначається локальною або глобальною дескрипторною таблицею за загальними правилами.

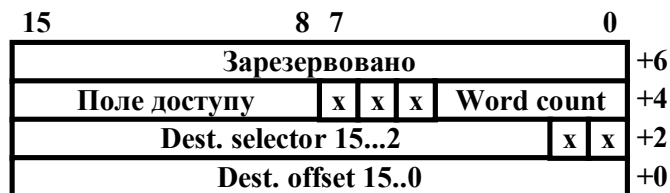


Рис. 18. Дескриптор шлюзу

Загальний формат дескриптора шлюзу зображений на рис. 18. На рисунку використані такі позначення: Dest. offset – зміщення початку процедури відносно початку сегмента, Dest. selector – селектор сегмента де розміщається код котрий викликається. Поле Dest. selector для шлюзів виклику, переривань та пасток задає селектор відповідного сегмента коду, а для шлюзів задач – селектор сегмента TSS. Поле доступу містить код прав доступу до нового сегменту коду, а Word count містить кількість байт які необхідно скопіювати у стек викликаної задачі.

Існують декілька типів шлюзів: шлюз виклику (Call Gate), шлюз задачі (Task Gate), шлюз переривання (Interrupt Gate) шлюз пастки (Trap Gate). Не усі шлюзи можуть використовуватись у всіх дескрипторних таблицях. Шлюз виклику не можна використовувати у IDT, а шлюзи переривань та пасток може використовуватись лише у IDT.

Не усі поля використовуються у шлюзах різного типу. Поле Word count використовується лише у шлюзах виклику, а поле Dest. offset не використовується в шлюзах задачі.

Шлюз задачі може використовуватись у будь-якій з дескрипторних таблиць. Особливість шлюза, – у якості сегмента указаний сегмент TSS. Поле зміщення не використовується та може мати будь-яке значення.

Особливістю шлюза переривань є скид ознаки дозволу переривань IF при вході у процедуру, що викликається. Попереднє значення регістра ознак

зберігається у стеку, тому при виході із процедури обробки ознака IF приймає попереднє значення.

Шлюз пастки, працює аналогічно шлюзу переривань, але замість ознаки IF скидає ознаку TF.

Алгоритм переходів з використанням шлюзів:

- ◇ Якщо рівень привілей викликаного сегмента, чисельно менше CPL, відбувається зміна стека, із відповідних полів TSS завантажуються нові значення SS, SP;
- ◇ Якщо була зміна стека, то в новий стек зберігається попереднє значення SS, SP;
- ◇ Якщо була зміна стека, а шлюз є шлюзом виклику, то в новий стек копіюються Word Count слів зі стеку попередньої програми;
- ◇ Якщо відбувається перехід на шлюз переривання чи пастки зберігається значення регістра F;
- ◇ Зберігається біжуче значення CS, IP;
- ◇ Із дескриптора шлюзу у регістри CS:IP завантажується нова адреса, задана відповідними полями дескриптора.

5.6. Віртуальна пам'ять

Процесор i80286 може безпосередньо адресувати до 16 Мбайтів фізичної пам'яті. Програми можуть вимагати для свого запуску об'єму пам'яті, що не існує у системі. Тоді операційна система може використати так звану віртуальну пам'ять.

Основна ідея віртуальної пам'яті полягає у тому, щоб зберігати (та оновлювати) вміст великої віртуальної пам'яті на диску, "підкачуючи" окремі ділянки віртуальної пам'яті в реальну оперативну пам'ять.

Можна, наприклад, зберігати усі використовувані програмою сегменти на диску, а у фізичну пам'ять записувати тільки сегменти, необхідні для виконання програми в даний момент.

Операційна система веде облік сегментів, що знаходяться в пам'яті та на

диску. Два поля дескриптора в полі доступу: біт присутності сегмента в пам'яті Р і біт звертання до сегмента пам'яті А призначені для апаратної реалізації обліку сегментів.

При використанні віртуальної пам'яті непотрібні у даний момент сегменти записуються на жорсткий диск, а звільнене місце використовується потрібними даними. У дескрипторі таких "непотрібних" сегментів, у полі Base указується розміщення сегмента на жорсткому диску, одночасно у полі доступу біт Р встановлюється в 1. Коли відбувається звертання до даних розміщених у сегменті з бітом $P = 1$ у полі доступу, викликається спеціальне виключення (#NP), процедура обробки котрого зчитує потрібні дані з пам'яті, змінює дескрипторні таблицю, та повторно виконує команду, котра звернулася до вказаних даних.

Контрольні питання

1. Мікропроцесор i80286: характеристики МП та режими роботи.
2. Мікропроцесор i80286: призначення та позначення виводів МП.
3. Регістри загального призначення та індексні регістри МП.
4. Сегментні регістри, їх використання у реальному режимі роботи.
5. Сегментні регістри, їх використання у захищеному режимі роботи.
6. Регістри LDTR, GDTR, F, MSW, TR, їх структура та використання.
7. Формування фізичної адреси команд та даних в захищеному режимі роботи МП i80286.
8. Формат селекторів та дескрипторів у МП i80286.
9. Організація мультизадачності мікропроцесора i80286. Сегмент стану задачі.
10. Принцип віртуальної пам'яті.
11. Шлюзи. Їх формат та використання.

Мікропроцесор i386

1. Основні характеристики

Мікропроцесор i386 є високопродуктивним 32-бітним процесором, що має 32-бітну архітектуру, 4 гігабайти фізичного адресного простору та внутрішнє забезпечення роботи зі сторінковою віртуальною пам'яттю. 32-бітна архітектура i386 забезпечує програмні ресурси, необхідні для підтримки "великих" систем, які характеризуються операціями з великими числами та структурами даних, великими програмами чи великою кількістю програм. Мікропроцесор i386 фірми Intel (або i80386 як його було названо у оригінальній документації) продовжує еволюцію 16-бітних процесорів i8086/i8088. У ньому збережена програмна сумісність з МП i8088, i8086 і i80286, що дозволяє використовувати програми, які розроблені для його попередників. Однак у процесорі i386 перейдено істотне обмеження в 64 кбайти на довжину неперервної структури даних. Довжина сегмента пам'яті у МП i386 збільшилась до 4 Гбайт. Крім того фізичний адресний простір i386 розширений до 4 Гбайт; а логічний адресний простір (доступний, наприклад, через зовнішні регістри сторінок) до 64 терабайт (2^{46} байт).

i386 сумісний з i80286 і з i8086 на рівні двійкового коду. Хоча можна просто використовувати i386 як швидкий i80286 чи як дуже швидкий i8086, їх сумісність гнучка. Процесор i386 може виконувати програми i8086, i80286 та свої 32-бітні паралельно, використовуючи захищений режим i386. Тобто за i386 може паралельно виконувати програми, написані для різних поколінь мікропроцесорів.

Мікропроцесор i386 має повну систему команд для операцій над різними типами даних, а також для керування виконанням програм. Способи адресації i386 забезпечують ефективний доступ до елементів стандартних структур даних: масивів, записів, масивів записів і записів, що містять масиви.

Важливішою властивістю сумісності i80386 з попередніми моделями є так званий, режим віртуального 86 процесора, що дозволяє використати будь-яке

програмне забезпечення i8086 у захищеному режимі роботи i386.

Мікропроцесор i386 розпізнає усі команди i80286, регістри, дескриптори і т.п., тобто операційна система i80286 і прикладні програми можуть бути перенесені на аналогічне устаткування, побудоване на основі i386 без змін.

З іншого боку операційні системи для i386 можна розробляти так, щоб вони підтримували існуючі прикладні програми для i80286, дозволяючи у той же час новим прикладним програмам використовувати властивості архітектури i386 (наприклад, 32-бітні параметри і довгі сегменти). У таких гнучких розробках нові прикладні програми викликають операційну систему безпосередньо, передаючи 32-бітні параметри. Виклик старих прикладних програм, які організовані в 16-бітному форматі i80286, перехоплюються і перетворюються у 32-бітний формат, а потім передаються в операційну систему.

2. Структура мікропроцесора i386

Структурна схема МП i386 зображена на рис. 19. Вона складається із пристрою обробки (нижня частина схеми), пристрою формування адреси пам'яті (верхня частина схеми) та пристрою узгодження з шиною (права частина схеми).

Пристрій обробки складається із блока АЛП, блока керування, блока декодера та блока черги команд. Блок черги команд складається із пристрою вибірки та 16-байтової черги команд. Пристрій вибірки зчитує команди з пам'яті та розміщає їх у 16-байтовій черзі команд, одночасно пристрій виконує контроль меж сегмента кода. Коди з черги команд надходять у блок декодера де за допомогою дешифратора команд відбувається перетворення коду у дешифрований код інструкції. Дешифровані коди накопичуються у 3-байтовій черзі. По мірі необхідності вони подаються на блок керування. Блок керування складається із керуючого ПЗП мікрокоманд та програмного автомата. На основі дешифрованого коду команди та керуючого ПЗП мікрокоманд програмний автомат формує сигнали керування блоками МП, у тому числі блоком АЛП.

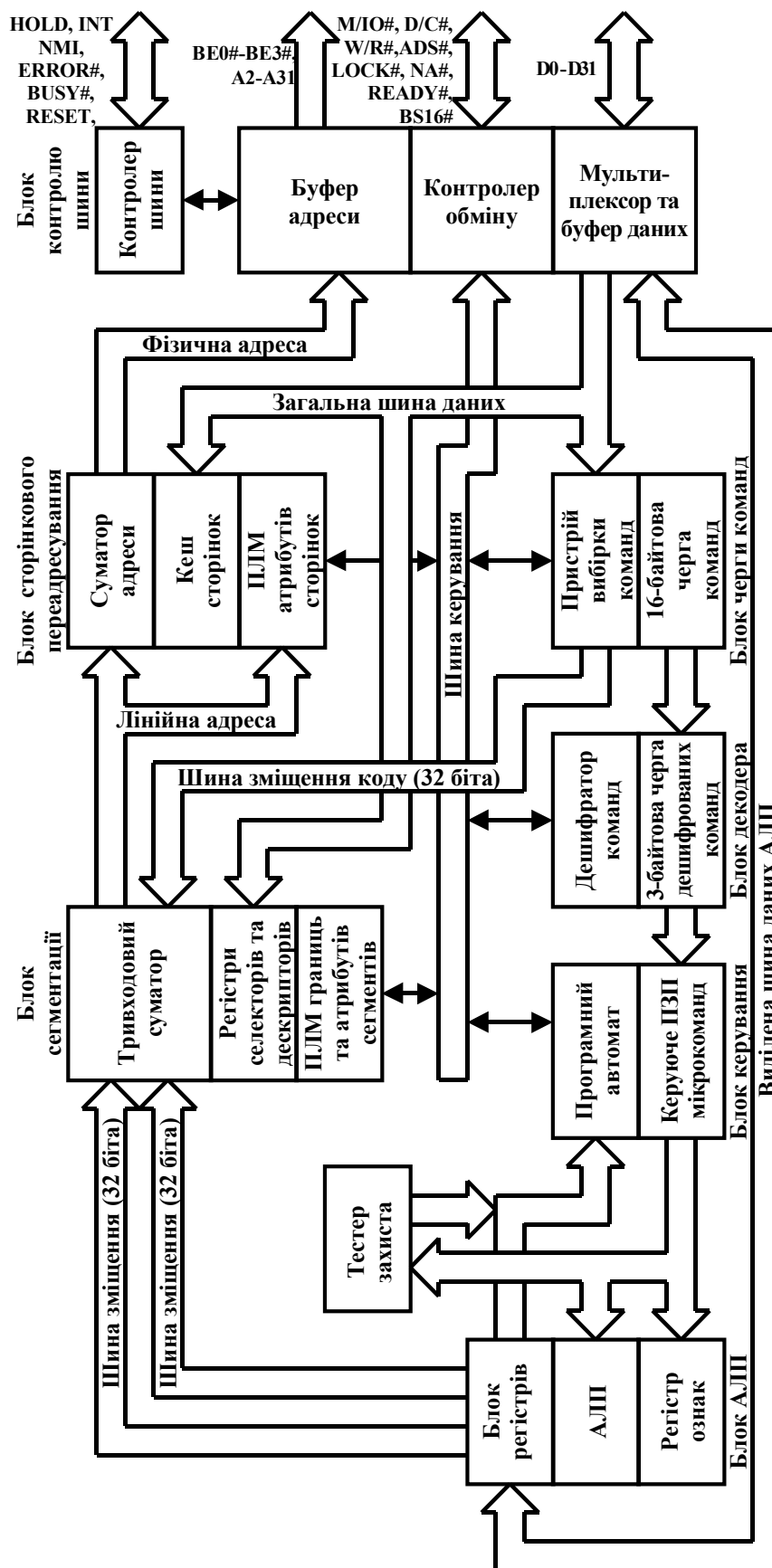


Рис. 19. Структура мікропроцесора i386

Блок арифметико-логічного пристрою містить вісім 32-бітних реєстрів

загального призначення об'єднаних у блок регістрів. Вони є взаємозамінними та використовуються у формуванні адреси у різних способах адресації, а також для збереження даних. Частина деяких регістрів МП можуть також використовуватись у програмі для збереження 8- чи 16-бітних чисел. Також у блок АЛП входить арифметико-логічний пристрій (АЛП), що виконує усі арифметичні та логічні операції та регістр ознак, котрий зберігає ознаки результату останньої арифметичної та логічної операції. З вищеназваними блоками зв'язаний блок тестування захисту, що перевіряє коректність адреси результатів та операндів.

Пристрій адресації пам'яті складається із блока сегментації та блока сторінкового переадресування. Блок сегментації, в свою чергу, складається із багатовходового суматора, призначеного для формування адреси, блока селекторних та дескрипторних регістрів та програмованої логічної матриці границь та атрибутів. Блок сторінкового переадресування складається із суматора адреси, кеша сторінок та програмованої логічної матриці контролю.

Блок інтерфейсу шини складається із контролера шини, котрий розподіляє по пріоритетах запити, що надходять на нього, буфера адреси, контролера обміну та мультиплексора-буфера даних. Мультиплексор-буфер даних з'єднаний виділеною шиною з блоком арифметико-логічного пристрою.

Згадані вище блоки мікропроцесора i386 працюють відносно автономно і паралельно. Усі внутрішні шини, що з'єднують ці блоки, мають розрядність 32 біти. Конвеєрна організація функціональних блоків у i386 допускає часове накладання виконання різних стадій команди і дозволяє одночасно виконувати деякі операції. Крім конвеєрного виконання усіх команд, у i386 деякі важливі операції виконуються спеціальними апаратними вузлами. Наприклад арифметико-логічний пристрій (АЛП) виконує множення 32-розрядних чисел за 9–41 такт синхронізації, залежно від кількості значущих цифр. Він може розділити 32-бітні операнди за 38 тактів (у випадку чисел без знаків) чи за 43 такти (у випадку чисел зі знаками).

Пристрій формування адреси також має конвеєрну структуру, з можливістю відключення непотрібних блоків. Операційна система, що керує

роботою МП, дозволяє системі реального часу обходити сторінкову переадресацію, або відключити сегментацію пам'яті. Завдяки конвеєрній організації МП має високу швидкодію, у наслідок чого його продуктивність зазвичай обмежується часом доступу до пам'яті.

Процесор був спроектований так, щоб з максимальною ефективністю використовувати як найбільш швидкодіючі статичні ОЗП, так і недорогі динамічні. Для звертання до швидкої пам'яті, i80386 може виконувати двотактний магістральний цикл. Звертання до повільної пам'яті та пристроїв вводу-виводу може використовувати конвеєрне формування адреси.

Унаслідок внутрішнього конвеєрного формування адреси при виконанні команди i80386, як правило, обчислює адреси й визначає наступний магістральний цикл під час поточного циклу обміну даними. Вузол конвеєрного формування адреси передає цю випереджувальну інформацію у підсистему пам'яті, дозволяючи, тим самим, одному банку пам'яті підготуватись до наступного циклу, у той час як інший банк виконує поточний магістральний цикл.

3. Призначення виводів мікропроцесора

Призначення виводів мікропроцесора наведено у таблиці 8.

Таблиця 8. Призначення виводів МП

Позначення	Тип	Функція
1	2	3
CLK2	Вхід	Загальний сигнал синхронізації мікропроцесора
D31 – D0	Вхід-вихід	Шина даних. Призначена для передачі даних, кодів команд та векторів переривань
A31 – A2	Вихід	Шина адреси. По лініям відбувається звертання до пам'яті та портів вводу-виводу. Сигнали A0 та A1 внутрішньої шини адреси сховані і визначаються станом ліній BE3 – BE0.
BE3 – BE0	Вихід	Сигнали дозволу передачі та приймання даних. Логічний нуль на лінії вказує, що відповідний байт на шині дані даних дійсний і бере участь в передачі даних
W/R#	Вихід	Записування/читання. Логічна 1 указує на цикл записування даних, нуль – читання
RESET	Вхід	Сигнал скиду процесора. Активний рівень – логічна 1

Продовження таблиці 8

1	2	3
D/C#	Вихід	Дані/команди. Логічна 1 указує, на обмін даними по шині, нуль – на підтвердження переривання, зупинку процесора або в цикл вибірки команди
M/IO#	Вихід	Пам'ять/ввід-вивід. Логічний нуль на лінії указує на обмін з портами вводу-виводу, логічна 1 – з пам'яттю
LOCK#	Вихід	Сигнал монопольного захоплення шин. Логічний 0 указує на заборону переривати обмін
ADS#	Вихід	Адреса дійсна. Логічний нуль на лінії вказує на наявність на лініях МП дійсної адреси та типу циклу
NA#	Вхід	Наступна адреса. Логічний нуль на вході вказує, що зовнішній пристрій сприйняв адресу, а мікропроцесор може видавати адресу наступного обміну
READY#	Вхід	Логічний нуль на вході вказує на готовність зовнішнього пристрою до обміну і можливість завершення циклу обміну
BS16#	Вхід	Вхід, вказує розрядність шини по якій відбувається обмін. Логічний 0 указує на можливість обміну лише 16 бітними словами
HOLD	Вхід	Вхід запиту прямого доступу до пам'яті. Логічна 1 указує на запит прямого доступу
HLDA	Вихід	Вихід підтвердження прямого доступу до пам'яті. Логічна 1 указує на підтвердження запиту
BUSY#	I	Сигнал зайнятості співпроцесора. Активність сигналу зупиняє процесор у випадку виконання ним команди роботи з співпроцесором
ERROR#	Вхід	Сигнал помилки співпроцесора, викликає виключення при виконанні команд роботи з співпроцесором
PEREQ	Вхід	Вхід запиту на передачу даних співпроцесору
INTR	Вхід	Вхід маскованих переривань. Логічна 1 указує на запит переривання
NMI	Вхід	Вхід немаскованих переривань. Логічна 1 указує на запит переривання
RESET	Вхід	Сигнал скиду процесора. Активний рівень – логічна 1

4. Регістри i386

Програмна модель МП i386 зображена на рис. 20. Сірим позначені регістри, що не використовуються у МП.

До складу МП входять вісім 32-бітних регістрів загального призначення, які можна використовувати для проведення обчислень і формування адреси

комірки пам'яті. Вони є розширенням відповідних 16-розрядних регістрів i8086, на що вказує приставка Е перед назвою кожного регістра.

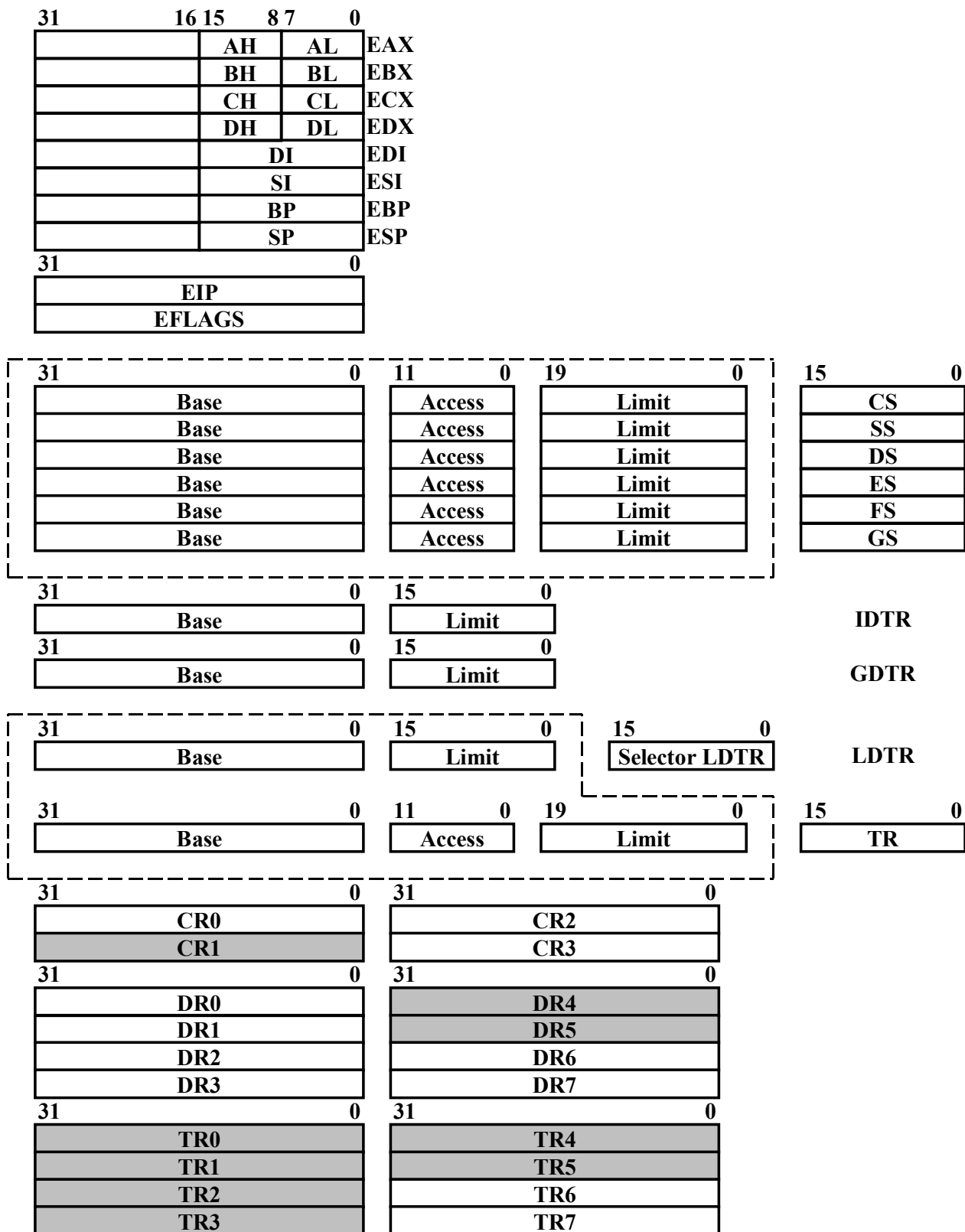


Рис. 20. Регістри МП i386

Виділяють регістри даних (EAX, EBX, ECX, EDX), а також базові та

індексні регістри (EBP, ESP, EDI, ESI). Є можливість звернутись до молодших 16 біт кожного 32-розрядного регістра загального призначення, для такого звертання необхідно використати назву регістра без префікса E. Наприклад команда **MOV EAX, EBX** запише у регістр EAX число, що зберігалось у регістрі EBX. А команда **MOV AX, 0** встановить у нуль 16 молодших біт числа в EAX, лишивши старші біти без змін. Нажаль, немає мнемоніки, котра дозволяла б так само легко оперувати старшими двома байтами 32-розрядного регістра. Однак, за допомогою команд зсуву, можна поміняти місцями 16 молодших і старших розрядів 32 розрядного регістра. Молодші 16 біт кожного з чотирьох регістрів даних (EAX, EBX, ECX, EDX) можуть бути використані у якості двох 8-бітних регістрів. Молодшій байт кожного з регістрів носить назву AL, BL, CL, DL, відповідно, а наступний AH, BH, CH та DH.

Чотири індексні регістри мікропроцесора i386 можуть адресуватись як 32-бітні (EBP, ESI, EDI, ESP), а також як 16-бітні (BP, SI, DI, SP). Ці регістри використовують також як адресні, або як операнди в командах обробки даних. Регістр ESP використовується у якості вказівника вершини стека.

Шість 16-розрядних сегментних регістрів (CS, SS, DS, ES, FS, GS) указують на початок відповідного сегмента пам'яті. Призначення регістрів CS, SS, DS, ES та відповідних сегментів таке ж саме, як у процесорі i80286. Сегментні регістри FS, GS використовуються як додаткові сегменти даних.

Лічильник команд EIP має розрядність 32 біти та призначений для визначення адреси наступної команди. Вміст лічильника інкрементується при виконанні кожного байта команди. При виникненні переривань та виключень, а також при виконанні команд передачі керування вміст лічильника команд перезавантажується новим значенням, а черга команд скидається.

Регістр ознак EFLAGS призначений для збереження ознак статусу та керування мікропроцесором. Ознаки поділяють на ознаки статусу та керування. Ознаки статусу визначають властивості результату виконання останньої арифметичної команди, команди маніпуляції бітами, або команди порівняння.

Наприклад, якщо два операнди при порівнянні виявляються рівними, то процесор встановлює ознаку нульового результату. Команди умовного переходу (та деякі інші), перевіряють ознаки статусу і дають різні результати залежно від їх стану.

Ознаки керування призначені для зміни виконання деяких команд, чи режимів роботи МП. Наприклад, команда перегляду рядка може мати напрямок до більших чи менших адрес залежно від стану ознаки напрямку. Формат регістра ознак зображено на рис. 21, призначення бітів регістра наведено у таблиці 9.

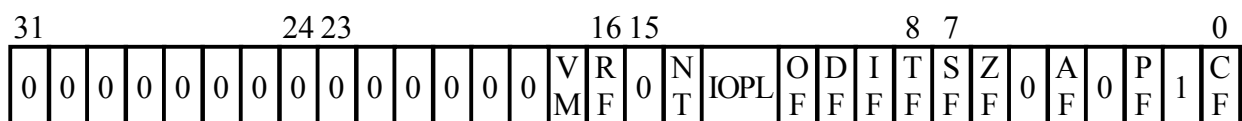


Рис. 21. Формат регістра ознак

Таблиця 9. Призначення бітів регістра ознак

Біт регістра ознак	Призначення
1	2
VM	Ознака режиму віртуального i8086. Біт може бути встановлений лише в захищеному режимі. У захищеному режимі встановлення біта в 1 вмикає режим віртуального i8086
RF	Ознака поновлення. Встановлення біта в 1 указує на поновлення "0" – нема помилки, "1" – помилка відлагодження. Ознака тимчасово вмикає засоби відлагодження, коли програма поновлює роботу після особливого випадку відлагодження
NT	Ознака вкладки задачі. "0" – поточна задача є вкладеною, "1" – поточна задача не є вкладеною
IOPL	Рівень привілеїв вводу-виводу. Два біти визначають один з 4-х рівнів. Стан "00" (нульовий рівень) є задає найбільш привілейований рівень, стан "11" (третій рівень) – найменш привілейований
OF	Ознака переповнення. Біт встановлюється, коли результат арифметичної операції не поміщається у заданій комірці
DF	Ознака напрямку операцій в командах обробки стрічок та послідовностей. Нуль указує на обробку з автоінкрементом (наступний операнд розміщений у комірках з більшими адресами), одиниця на обробку з автодекрементом (наступний операнд розміщений у комірках з меншими адресами)

Продовження таблиці 9

1	2
IF	Дозвіл переривання. Одиничне значення біта дозволяє обробку запиту переривання по лінії INT. Нульове значення забороняє її
TF	Ознака трасування. Коли біт установлений у стан 1 після виконання кожної команди відбувається переривання типу INT 1
SF	Ознака знака. Одиничне значення біта указує на від'ємний результат останньої арифметичної або логічної операції. Фактично при встановленні знака МП вважає усі числа знаковими та поданими у додатковому коді. Знак визначається старшим бітом результату
ZF	Ознака нуля. Одиничне значення біта указує на нульовий результат останньої арифметичної або логічної операції
AF	Ознака додаткового перенесення. Одиничне значення біта указує на перенесення із третього у четвертий біт при виконанні останньої арифметичної операції. Ознака використовується при виконанні
PF	Ознака парності. Одиничне значення указує, що у молодшому байті результату парне число одиниць, нульове – непарне
CF	Ознака перенесення. Одиничне значення указує на наявність перенесення із старшого біта результату

Розмір та розміщення глобальної дескрипторної таблиці визначається 48-розрядним регістром GDTR. Формат регістра зображено на рис. 22.

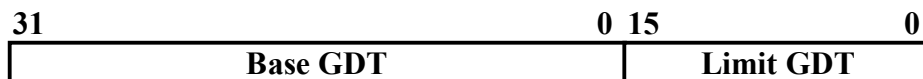


Рис. 22. Формат регістра GDTR

Поле Base GDT задає початкову лінійну адресу глобальної дескрипторної таблиці. Поле Limit GDT задає її довжину у байтах.

Розмір та розміщення локальної дескрипторної таблиці визначається 16-розрядним регістром LDTR, в якому зберігається селектор для вибору дескриптора локальної дескрипторної таблиці (LDT). Для прискорення роботи з локальною дескрипторною таблицею її дескриптор зберігається у "схованому" регістрі. Інформація у схованому регістрі перезавантажується при першому ж звертанні до LDT після виконання команди завантаження у LDTR.

Таблиця векторів переривань мікропроцесора i386 може бути розміщена будь-де у пам'яті. МП використовує 40-розрядний регістр IDTR для адресації

таблиці векторів переривання IDT, формат та призначення якого схожі на регістр GDTR (рис. 23).

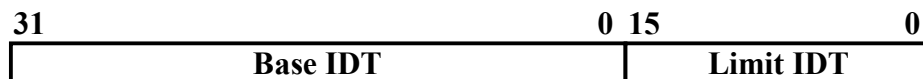


Рис. 23. Формат регістра IDTR

Для організації багатозадачної роботи мікропроцесора в захищеному режимі використовують сегмент стану задачі TSS, котрий зберігає копії усіх регістрів МП. Визначення положення сегмента стану задачі здійснюється 16-рядним регістром задачі TR, який містить селектор сегмента стану задачі TSS.

Наступна група регістрів є розширенням регістра MSW МП i80286 та призначена для керування роботою МП. У групу входять регістри CR0–CR3. Регістр CR0 забезпечує загальне керування режимами роботи процесора. Його формат зображений на рис. 24.

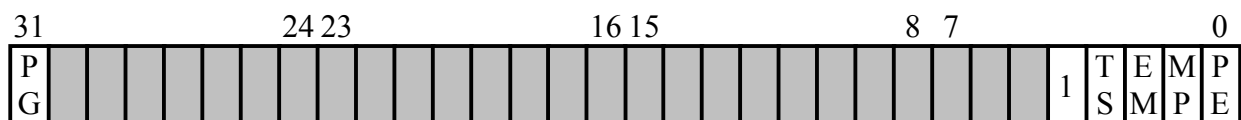


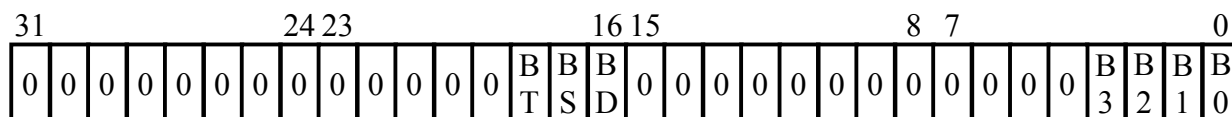
Рис. 24. Формат регістра CR0

Біти зображені сірим кольором у i386 не використовувались. Для узгодження зі старшими моделями у ці біти повинні бути записані нулі. Призначення бітів регістра CR0 наведено у таблиці 10.

Таблиця 10. Призначення бітів регістра CR0

Біт CR0	Призначення
PE	Дозвіл захисту. Одиничне значення вмикає захищений режим роботи, нульове переводить МП у реальний режим
MP	Моніторинг співпроцесора. Встановлення біта в 1 дозволяє викликати виключення #NM "співпроцесор не існує" при першій ж інструкції співпроцесора після переключення задач
EM	Емуляція співпроцесора. Встановлення біта в 1 дозволяє викликати виключення #NM при всіх інструкціях співпроцесора. Співпроцесор емулюється програмно
TS	Перемикання задач. Одиничне значення біта вказує на перемикання задач. Біт використовується для визначення стану співпроцесора
PG	Дозвіл роботи блока сторінкової переадресації. Одиничне значення біта дозволяє роботу блока сторінкової переадресації. Нульове значення забороняє його роботу

Регістр CR1 у МП i386 відсутній. Регістр CR2 зберігає 32-бітну лінійну адресу, за котрою була отримана остання відмова сторінки пам'яті. Регістр CR3 в старших 20 розрядах зберігає фізичну адресу таблиці каталогів сторінок.



Усі засоби відлагодження оперують перериванням INT01 (Exception 01). У регістрі DR6 зберігається причина виклику INT01. Формат регістра DR6 зображено на рис. 25. У 32-бітному регістрі DR6 використовуються лише 7 біт. Їх призначення наведено у таблиці 11.

Біт DR6	Призначення
BT	Переривання відлагодження викликане перемиканням задачі, в TSS якої біт T =1
BS	Переривання відлагодження викликане встановленим бітом TF=1. Коли встановлений біт BS, можуть бути встановлені і інші біти регістра
BD	Наступна команда буде зчитувати або записувати дані в один з регістрів відлагодження
B0 - B3	Відбулося звертання до адреси заданої в регістрах DR0-DR3

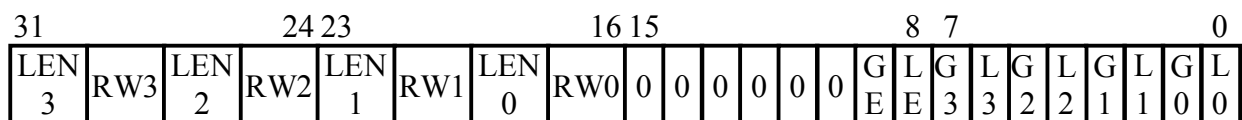


Рис. 26. Формат регістра DR7

Таблиця 12. Призначення розрядів регістра DR7

Біт DR7	Призначення
LEN _i	Поле визначає довжину структури даних. Переривання INT 01 відбувається якщо хоча б один байт буде зчитаний або записаний зі структури, що контролюється. Поле може приймати значення 1,2 або 4
WR _i	Властивість точки зупинки. Визначає тип звертання при котрому відбудеться переривання INT 01. Якщо WR _i = 00 переривання відбувається при виконанні кода команди, якщо WR _i = 01 – у випадку записування даних у комірку пам'яті, WR _i = 11 – у випадку читання або записування даних. Стан 10 вважається забороненим
GE, LE	Точність контролю. Встановлення GE або LE у 1 (для всіх або для одної задачі відповідно) сповільнює внутрішні операції процесора для своєчасної генерації INT 01. Необхідність сповільнення викликана тим, що із-за конвеєризації контрольна точка у області даних фіксується тільки після виконання декількох наступних команд, що затримує відлагодження
Gi	Глобальна точка зупинки, дозволена для всіх задач
Li	Локальна точка зупинки, при переключенні задач ця точка зупинки діяти не буде

Тестові регістри призначені для тестування сторінкової переадресації. Регістр TR6 використовується як регістр команд та адреси, регістр TR7 – як регістр даних. Через регістр TR7 можна переглянути роботу блока сторінкової переадресації та вплинути на його роботу. Регістри доступні у реальному режимі та у захищеному режимі при рівні привілеїв 0.

5. Переривання

Пристрої генерують переривання, коли вони вимагають уваги, в той час як команди можуть викликати особливі ситуації, якщо при їхньому використанні виникають особливі умови, такі – як неіснуюча сторінка. Типове переривання чи особлива ситуація вимагають швидкого втручання програмного драйвера, який відповідає на переривання чи особливу ситуацію. Після того, як драйвер поверне керування, МП і386 поновлює виконання програми, яка була перервана

чи викликала особливу ситуацію. Оскільки переривання й особливі ситуації дуже схожі, процесор i386 розглядає їх уніфікованим способом.

Так само як у i80286 у МП i386 є декілька типів переривань: програмні переривання, апаратні зовнішні переривання, апаратні внутрішні переривання та виключення. Кожне джерело переривання і кожен тип особливої ситуації має ідентифікаційний номер у діапазоні від 0 до 255.

Програмні переривання, це швидкий метод виклику підпрограм операційної системи. Програмні переривання поділяють на переривання за відлагоджувальними точками, за переповненням та за вектором.

Переривання за відлагоджувальною точкою генеруються за однобайтовою командою INT3. Основна задача цієї команди – дати можливість зупинити програму, що відлагоджується у будь-якому місці програми.

Переривання за переповненням генерується за командою INTO. Якщо ознака переповнення OF рівна 1 у момент виконання команди, то за командою здійснюється перехід до обробки переривання за вектором (номером) 4.

Переривання за вектором дозволяє викликати будь-яке переривання. Якщо у процесі виконання програми у ній зустрінеться команда INT x, де x - число від 0 до FFh, то відбудеться переривання з номером x.

Апаратні зовнішні переривання виникають при надходженні сигналу на вхід запиту переривання. У МП i386 також як і у i80286 є два входи запиту переривання: запит маскованих переривань INT та запит немаскованих переривань NMI. Масковані переривання можуть бути заблоковані скиданням біта дозволу переривань IF у регістрі ознак МП. При надходженні на вхід INT логічної одиниці, якщо переривання дозволені, здійснюється процедура підтвердження переривання. У процесі виконання підтвердження переривання пристрій, котрий запросив переривання, передає МП вектор переривання. Немасковані переривання виникають завжди при переході лінії NMI у стан логічної 1. Запити переривань сприймаються та обробляються після закінчення виконання команди.

До внутрішніх апаратних переривань відносять переривання по помилці

ділення, переривання покрокової роботи та переривання за апаратними точками відлагодження. Переривання по помилці ділення відбувається після виконання команди ділення (DIV або IDIV), якщо результат ділення перевищує по модулю максимальну допустиму величину. Покрокове переривання призначено для відлагодження програми, воно відбувається після виконання кожної команди програми, якщо біт TF регістра ознак рівний 1. Переривання за апаратними точками відлагодження відбувається, якщо адреса та тип звертання до пам'яті співпадає з указаним у регістрах відлагодження. Усі згадані переривання можуть бути викликані у реальному та захищеному режимах роботи.

Так само як у процесорі i80286 у МП i386 є особливий клас переривань, що носять назву "виключення". На відміну від "класичних" внутрішніх апаратних переривань, виключення можуть бути оброблені до закінчення виконання команди, що спричинила появу виключення. Усі виключення розділяють на три класи:

- ◇ Відмова (fault) – виключення, що виявляється та обслуговується до закінчення виконання команди, котра викликала виключення. Після виконання підпрограми обслуговування відмови можливе повторне виконання команди.
- ◇ Пастка (trap) – виключення, що виявляється та обслуговується після команди, що його викликала.
- ◇ Аварія (abort) – виключення, що не дозволяє точно встановити команду, котра його викликала. Після завершення обслуговування такого виключення подальше виконання програми неможливе.

Множина виключень залежить від режиму роботи МП. Інформація про усі виключення та переривання зображені у таблиці 13. У таблиці використовуються позначення: F - виключення типу "відмова"; A - виключення типу "аварія"; T - виключення типу "пастка"; AI - зовнішнє апаратне переривання; PI - програмне переривання; ЕС - перед викликом підпрограми обробки виключення у стек заноситься ідентифікаційний код виключення, ЕС = 0 - перед викликом підпрограми обробки виключення у стек заноситься код 0000h.

Таблиця 13. Переривання та виключення

Функція	Номер вектора	Тип
Переривання та виключення, наявні у реальному та захищеному режимах		
Переривання за помилкою ділення (#DE)	0	F
Покрокове переривання, та переривання за апаратними відлагоджувальними точками (#DB)	1	T
Немасковане переривання	2	AI
Переривання за відлагоджувальною точкою (#BP)	3	T
Переривання за переповненням (INTO) (#OF)	4	T
Виключення за контролем діапазону BOUND (#BR)	5	F
Недопустимий код операції (#UD)	6	F
Співпроцесор недоступний виконуваний програмі (#NM)	7	F
Апаратні масковані переривання. Номер вектора X передається у другому циклі підтвердження переривання.	X	AI
Програмні переривання за командою INT X	X	PI
Співпроцесор звертається до даних за межами сегмента	9	F
Переривання та виключення, наявні лише у реальному режимі		
Вектор відсутній у таблиці векторів переривань	8	A, EC=0
Сегмент стека переповнений, або спустошений (#SS)	0Ch	F, EC
Процесор звертається до даних або коду за межами сегмента (#GP)	0Dh	F, EC
Переривання та виключення, наявні лише у захищеному режимі		
Подвійне виключення (#DF)	8	A, EC=0
Недійсний сегмент стану задачі (#TS)	Ah	F, EC
Сегмент кода, або даних відсутній у пам'яті (#NP)	0Bh	F, EC
Сегмент стека переповнений, переспустошений, або відсутній (#SS)	0Ch	F, EC
Загальна відмова захисту (#GP)	0Dh	F, EC
Відмова сторінки (#PF)	0Eh	F, EC
Резерв	0Fh	–
Виключення співпроцесора (#MF)	10h	F

Фірма Intel зарезервувала переривання з номерами 0 – 20h (0 – 31) для подальшого використання у старших моделях процесорів, проте у реальному режимі роботи більшість цих векторів використовується операційними системами реального режиму роботи. Наприклад у ПК вектор виключення співпроцесора (#MF) використовується процедурами виводу зображення. Операційні системи захищеного режиму дозволяють оминати таку проблему.

У випадку виникнення переривання або виключення у залежності від його типу формується номер. Номери векторів переривань наведені у таблиці 13. По номеру вектора здійснюється пошук адреси підпрограми обробки переривання у таблиці векторів переривань IDT.

Формат IDT близький до формату таблиці векторів переривань МП i8086, у таблиці по послідовних адресах у реальному режимі роботи розміщені зміщення та сегмент відповідної підпрограми обробки переривання. У захищеному режимі роботи на позиції сегмента розміщений селектор, котрий посилається на шлюзи задач (Task Gate), шлюзи переривань (Interrupt Gate), або шлюзи пасток (Trap Gate), а сам обробник переривань чи особливих ситуацій може бути реалізований у вигляді процедури чи задачі.

Таблиця IDT може розміщатись будь-де у пам'яті. Для встановлення початкової адреси таблиці векторів переривань МП використовує 4-розрядний регістр IDTR, формат та призначення розрядів котрого схожі на формат регістра GDTR.

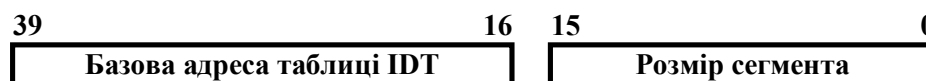


Рис. 27. Формат регістра IDTR

6. Реальний режим роботи

Реальний режим роботи призначений для забезпечення найбільшої сумісності з МП i8086. Адресація пам'яті у реальному режимі роботи здійснюється аналогічно до i8086, ніякого захисту програм та даних немає.

У реальному режимі процесор забезпечує швидке виконання програм без захисту так, як і на i8086. Принципова відмінність між реальним режимом i386 і процесором i8086 полягає у швидкості: програми i8086, працюють на i386 значно швидше.

Для адресації пам'яті МП використовує логічні та фізичні адреси. Логічна адреса є 32-бітною незалежно від режиму роботи МП. Вона містить 16-розрядний сегмент та 16-розрядне зміщення. Логічну адресу звичайно записують у формі: сегмент:зміщення. У реальному режимі мікропроцесор

звертається до пам'яті, генеруючи 20-розрядні (точніше, "майже 20-розрядні") фізичні адреси.

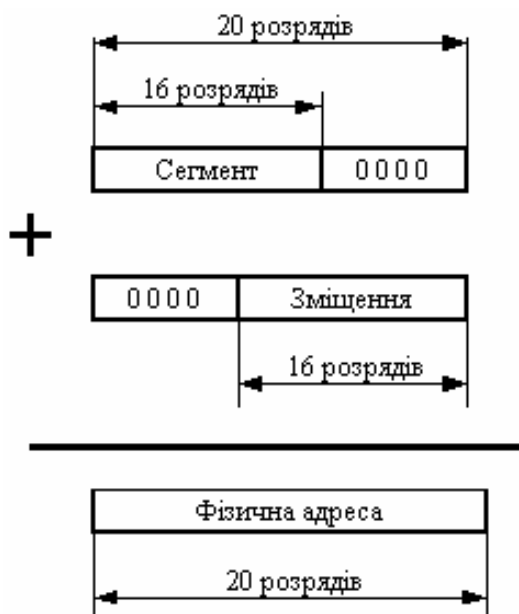


Рис. 28. Формування фізичної адреси у реальному режимі роботи

Формування фізичної адреси здійснюється по правилу зображеному на рис. 28: сегментна частина зсувається на 4 розряди вліво та додається до зміщення. Отже адреси початку сегментів завжди починаються з числа, кратного 16, а кожен сегмент пам'яті має розмір 64 кбайти. Легко довести, що одній фізичній адресі відповідає 16 384 різних логічних адрес. Наприклад: фізичній адресі 00400h (0000 0000 0100 0000 0000b) відповідають логічні адреси 0040:0000 та 0000:0400.

Так само як у i80286 розробники МП i386 лишили можливість адресувати дані за межами першого мегабайта. Для цього необхідно, фізична адреса, що формується за схемою рис. 28, стала більшою 0FFFFFFh. Наприклад, при значенні логічної адреси 0FFFFFFh:0010h відбудеться звертання по фізичній адресі 100000h, котра знаходиться за межами першого мегабайта. Фактично у реальному режимі наявний ще один додатковий сегмент пам'яті, що лежить вище межі першого мегабайта. Цей сегмент називається областю верхньої пам'яті (High Memory Area). Йому відповідає діапазон логічних адрес від 0FFFFFFh:0010h до 0FFFFFFh:0FFFFFFh. Розмір сегмента складає 64 кбайта без 16 байтів.

У МП i386 як і у i80286 існує можливість оминати межу в 1 Мбайт на розмір використовуваної пам'яті у реальному режимі роботи. Наприклад, можна встановити межу сегмента рівною 0FFFFFFFh та адресувати будь-яку комірку пам'яті у межах усього адресного простору МП. Також можна встановити базову адресу одного з сегментів даних за межами першого мегабайта. Вказані дії можна здійснити за допомогою відлагоджувальної команди LOADALL, котра завантажує більшість регістрів МП.

У реальному режимі роботи будь-яка програма може встановити довільні значення в сегментні регістри і адресувати будь-яку ділянку пам'яті. Зокрема, будь-яка програма може навмисно чи через помилку зруйнувати області даних, що належать операційній системі.

У реальному режимі передавання керування виконується за допомогою команд JMP, CALL, INT, RET, IRET, а також при виникненні переривань. При внутрисегментньому передаванні керування у регістр IP заноситься нове значення, а регістр CS не модифікується. Міжсегментне передавання і одночасно змінює регістри CS і IP, а також у деяких випадках і регістр ознак F (переривання і команди RET, IRET).

У реальному режимі будь-якій програмі дозволяється робити перехід на будь-яку адресу, або викликати будь-які підпрограми та переривання. Є можливість здійснити перехід на команди початкової ініціалізації системи або спробувати виконати дані. Єдина умова для успішного виклику підпрограми – знання адреси підпрограми (сегмента і зміщення) і формату переданих даних.

Отже два основні недоліки реального режиму:

- ◇ обмежений адресний простір;
- ◇ вільний доступ для будь-яких програм, до будь-яких областей даних, що становить потенційну небезпеку для цілісності операційної системи.

Цих недоліків повністю позбавлена схема адресації пам'яті, яка використана у захищеному режимі.

7. Захищений режим

У захищеному режимі роботи МП i386 використовують три типи адресації:

- ◇ логічна адреса – котра складається із сегмента та зміщення. Її використовують аналогічно до логічної адреси МП i80286;
- ◇ лінійна адреса – 32-розрядна адреса, котра формується на основі сегмента та зміщення у захищеному режимі роботи. Адреса використовується блоком сторінкової переадресації;
- ◇ фізична адреса – адреса, котра формується на основі лінійної блоком сторінкової переадресації та видається на лінії шини адреси.

Захищений режим роботи МП є основним режимом роботи МП. У цьому режимі процесор дозволяє адресувати до 4 Гбайт фізичної пам'яті. Для адресації пам'яті використовують чотири моделі розбиття адресного простору МП: лінійну, сегментну сторінкову та сторінково-сегментну.

У лінійній (плоскій) моделі логічний адресний простір складається з масиву байтів, що не має визначеної структури. Трансляції адреси в цьому випадку не потрібна, оскільки логічна адреса еквівалентна лінійній і фізичній адресі.

У сегментній моделі логічний адресний простір складається із сегментів, кожен з яких містить змінне число байтів. Логічна адреса складається з номера сегмента і зміщення всередині сегмента. Цей режим широко використовується у МП i80286.

У сторінковій моделі логічний адресний простір складається з набору сторінок, кожна з яких містить фіксовану кількість байтів. Логічна адреса складається з номера сторінки і зміщення усередині сторінки.

У сторінково-сегментній моделі логічний адресний простір складається із сегментів, які, у свою чергу, складаються зі сторінок. Логічна адреса складається з номера сегмента і зміщення усередині сегмента.

У МП i386 можливе використання усіх чотирьох моделей представлення адресного простору.

7.1. Сегменти

Сегмент це виділена область адресного простору певної довжини. Сегменти є логічними блоками, які добре пристосовані під програмні структури, що мають змінну довжину. Маючи для сегментів відповідні архітектурні рішення, i386 підвищує продуктивність систем, у яких механізм структурування побудований на сегментах. Сегмент у захищеному режимі МП i386 може мати будь-який розмір від 1 байта до 4 Гбайтів.

Логічний адресний простір задачі складається з одного чи декількох сегментів. Команди i386 посилаються на операнд у пам'яті за допомогою логічної адреси сегмента, що складається із номера сегмента та зміщення у сегменті. У захищеному режимі роботи номер сегмента, називають селектором.

У захищеному режимі роботи вміст сегментного реєстра призначений для пошуку базової адреси у спеціальній таблиці базових адрес, що розміщена у пам'яті. Таку таблицю називають дескрипторною, її рядок складається з 64 бітів і носить назву дескриптора. Для кожного сегмента операційна система підтримує заданий дескриптор, що містить опис сегмента. Опис сегмента містить 32-бітну базову адресу сегмента у лінійному адресному просторі, довжину сегмента, а також інформацію про захист, що запобігає неправильному використанню. Додавання зміщення до базової адреси дає лінійну адресу.

У МП i386 трансляція логічної адреси в лінійну оптимізована за допомогою неявної вказівки на селектори і збереження дескрипторів у схованих реєстрах МП.

7.2. Селектори

Селектор указує номер та властивості сегмента у захищеному режимі роботи. Формат селектора зображено на рис. 29.

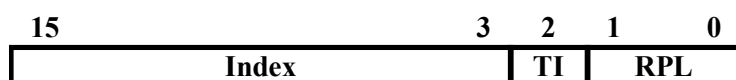


Рис. 29. Формат селектора

Два молодші розряди селектора (RPL), визначають чотири рівні

привілеїв, що запитує програма при доступі до сегмента. Привілеї дозволяють використовувати одні й ті ж самі сегменти для різних програм, або у різних місцях програми по різному.

Наступний біт селектора (TI) указує тип дескрипторної таблиці. Мікропроцесор i386 як й i80286 використовує дві дескрипторні таблиці:

- ◇ глобальну дескрипторну таблицю GDT, яка адресує сегменти, доступні будь-якій задачі;
- ◇ локальну дескрипторну таблицю LDT, що адресує сегменти, доступні лише конкретній задачі.

Якщо біт TI рівний 0 вибирається глобальна дескрипторна таблиця, якщо 1 – локальна дескрипторна таблиця.

Старші тринадцять розрядів селектора (Index), вказують на номер 64-розрядного запису з інформацією про базову адресу, розмір сегмента та права доступу. Зазвичай нульове значення селектора не використовується.

7.3. Дескрипторні таблиці

Кожна задача може мати доступ до системних та індивідуальних сегментів, ці сегменти описує глобальна таблиця дескрипторів (GDT) і локальна таблиця дескрипторів (LTD), відповідно. Таблиці GDT та LTD можуть містити до 8192 дескрипторів кожна, що спільно визначають логічний адресний простір задачі. Прикладні програми самі не мають права змінювати стан дескрипторних таблиць. Щоб новий сегмент став доступним програмі, операційна система повинна внести його дескриптор у GDT, або LTD.

Глобальна таблиця дескрипторів, доступна для всіх задач. Операційна система заносить дескриптори системних сегментів у GDT разом із сегментами, що мають бути доступні усім задачам.

Кожна задача може мати свою локальну таблицю дескрипторів. Процесор зберігає адресу LDT поточної задачі у своєму регістрі локальної таблиці дескрипторів (LDTR), та перезавантажує його новим значенням при переключення задач. Задачі можуть спільно використовувати одні й ті ж

сегменти якщо виконуються умови:

- ◇ Deskриптор сегмента знаходиться у GDT. Такий сегмент спільний для усіх задач.
- ◇ Задачі, мають спільну LDT. У цьому випадку усі сегменти задач спільні. Такий підхід припустимий у випадку задач, що тісно взаємодіють.
- ◇ Deskриптори у різних LDT вказують на один і той же сегмент. Такий сегмент буде спільним для обох задач. Такі deskриптори називають псевдонімами. Використання псевдонімів дозволяє задачам мати спільними тільки потрібні сегменти.

7.4. Deskриптори

Deskриптори призначені збереження інформації про сегменти даних, коду та сегменти службового призначення. Deskриптори сегментів коду, даних та деяких службових сегментів (deskриптори шлюзів та задач) поміщають у deskрипторну таблицю.

У deskрипторі задається рівень привілеїв, що може отримати програма при доступі до сегменту. Крім того у deskрипторі задається тип сегмента, його доступність, а також властивості сегмента. Наявність у deskрипторах сегментів прав доступу забезпечує захист від запису у сегменти коду, заборону виконання сегменту даних, контроль за переповненням сегментів стеку, тощо.

Усі deskриптори можна розділити на deskриптори даних, deskриптори коду та службові deskриптори. Незалежно від призначення deskриптори мають формат зображений на рис. 30.

15	8	7	0	
Base 31...24		Properties	Limit 19..16	+6
Access		Base 23...16		+4
Base 15...0				+2
Limit 15...0				+0

Рис. 30. Формат deskриптора сегмента

Тип deskриптора визначається бітами у байті поля доступу (Access).

Межа сегмента визначається полем Limit (Limit 19..16, Limit 15..0), вона визначає максимальне або мінімальне зміщення котре може бути використане при адресації даних. Фізична адреса початку сегмента визначається полем Base (Base31..24, Base23..16, Base15..0). 4 біти поля Properties задають властивості сегмента.

На рис. 31 наведені формати поля Access усіх типів дескрипторів: дескрипторів сегментів коду, сегментів даних і системних сегментів.

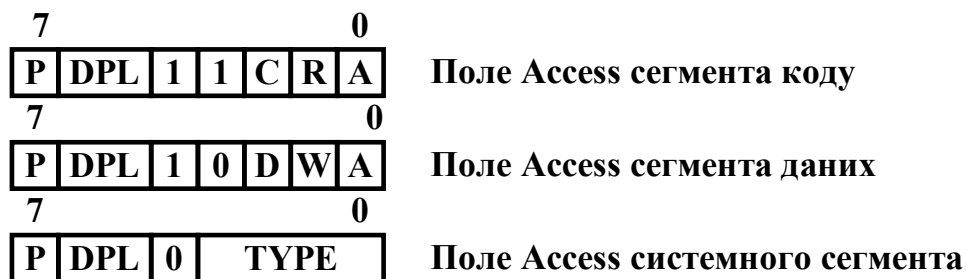


Рис. 31. Формат поля Access

У всіх дескрипторів три старших біти поля доступу мають однакове призначення. Біт P призначений для організації віртуальної пам'яті. Він указує на присутність сегмента в пам'яті. Для тих сегментів, що знаходяться у пам'яті, цей біт повинен бути встановлений в 1. Будь-яка спроба програми звернутися до сегмента пам'яті, у дескрипторі якого біт P встановлений в 0, призведе до виключення (#NP).

Поле DPL задає атрибути привілеї сегмента. Поле DPL контролює, коли сегмент може використовуватись задачею.

Четвертий біт визначає властивість сегмента. Якщо сегмент є сегментом даних, або коду біт рівний 1, якщо ж сегмент є системним - біт рівний 0.

Загальним для усіх типів сегментів є також біт A (молодший біт поля доступу), котрий називають бітом доступності сегмента. Для всіх сегментів, до котрих було звертання, або виконувалась команда тестування селектора біт автоматично встановлюється в 1. У випадку, коли звертань до сегмента ще не було біт установлюється в 0.

Третій біт поля доступу сегментів коду та даних дозволяє розрізнити сегменти даних та коду. У сегментів коду біт рівний 1, у сегментів даних – 0.

Біт С поля доступу сегментів коду визначає підпорядкованість сегментів. Якщо $C = 0$, то код з цього сегмента може виконуватись лише коли $DPL = CPL$. Такі сегменти називаються непідпорядкованими. У випадку коли $C = 1$, код з цього сегмента може виконуватись для всіх задач, якщо виконується умова $CPL \geq DPL$. У такому випадку код буде виконуватись з рівнем привілеїв, що задавався бітами CPL, а не DPL.

Поле доступу дескриптора сегментів коду містить бітове поле R, яке називають бітом дозволу читання сегмента. Якщо цей біт встановлений в 1, програма може зчитувати вміст сегмента коду. У протилежному випадку процесор може тільки виконувати код.

Дескриптори сегмента даних мають бітові поля W і D. Поле W називають бітом дозволу записування даних в сегмент. Якщо цей біт встановлений в 1, поряд з читанням можливе і записування даних у даний сегмент. В іншому випадку при спробі читання виконання програми буде перервано.

Поле D задає напрямок розширення сегмента. Звичайний сегмент даних розширюється в область старших адрес (розширення вгору). Якщо ж у сегменті розміщений стек, розширення звичайно відбувається у зворотному напрямку – в область молодших адрес (розширення вниз). Для сегментів, у яких організуються стек, необхідно встановлювати поле D в 1.

Дескриптори системних сегментів у чотирьох молодих бітах містять поле TYPE, що визначає тип системного сегмента. У таблиці 14 наведені можливі для цього поля значення.

Дескриптори службових сегментів, наприклад, які описують розташування таблиці LDT чи сегмента стану задачі, не можна використовувати для читання чи запису, навіть якщо програма виконується у нульовому пріоритетному кільці. У випадку необхідності можна створити додаткові дескриптори, у яких ці ж сегменти описані як сегменти даних. Так само слід поступати, якщо необхідно виконати запис у сегмент коду.

Поле Properties містить 4 біта та визначає властивості сегмента. Формат поля Properties указаний на рис. 32.

Таблиця 14. Типи системних сегментів

Поле TYPE	Тип сегмента
0000	Заборонене значення
0001	Дескриптор TSS доступної задачі у стилі i80286
0010	Сегмент LDT
0011	Дескриптор TSS зайнятої задачі у стилі i80286
0100	Вентиль виклику у стилі i80286
0101	Вентиль задачі у стилі i80286
0110	Вентиль переривання у стилі i80286
0111	Вентиль пастки (trap) у стилі i80286
1000	Заборонене значення
1001	Дескриптор TSS доступної задачі у стилі i386
1010	Зарезервовано
1011	Дескриптор TSS зайнятої задачі у стилі i386
1100	Вентиль виклику у стилі i386
1101	Вентиль задачі у стилі i386
1110	Вентиль переривання у стилі i386
1111	Вентиль пастки (trap) у стилі i386

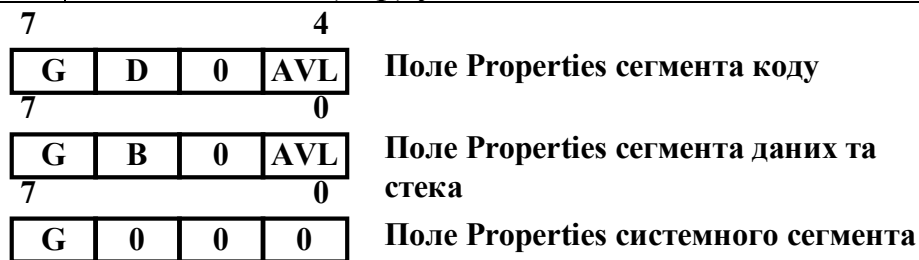


Рис. 32. Формат поля Properties

Поле G – біт гранулярності. Визначає одиницю виміру сегментів. Якщо G=0, то довжина сегмента визначається у байтах, якщо G=1 довжина визначається у сторінках по 4 кбайти, а довжина сегмента може сягати 4 Гбайт.

Поле D визначає розрядність зміщення та даних. Якщо D = 0 МП по замовчуванню оперує 16 розрядними даними та адресами, у протилежному випадку 32-розрядними. Розрядність зміщення та даних може бути змінена за допомогою префікса перекриття розрядності даних та зміщення.

Поле B визначає максимальну границю сегмента. При B = 0 максимальна границя сегмента рівна 0FFFFh, якщо B = 1 максимальна границя сегмента 0FFFFFFFFh. У сегментах стека біт B також визначає розрядність стека. Якщо B = 0, МП використовує 16 розрядний стек, команди POP та PUSH працюють з

16 бітними числами. Якщо $B = 1$, МП використовує 32 розрядний стек, команди POP та PUSH працюють з 32 бітними числами.

Поле AVL не використовується МП і доступне для змін операційній системі. Операційна система може використовувати поле за своїм розсудом для збереження властивостей сегмента. Наприклад, може зберігатись інформація про можливість витіснення сегмента на жорсткий диск у процесі підкачки.

7.5. Формування лінійної адреси

В будь-який момент часу програма може звертатись до одного із 6 сегментів, що відповідають одному з 6 сегментних регістрів. Кожний тип даних, що використовує МП має свій сегментний регістр, який визначає селектор цих даних. Можливість використання сегментних регістрів з різними типами даних указана у таблиці 15.

Таблиця 15. Можливість використання сегментів.

Дані що адресуються	Сегмент					
	CS	DS	SS	ES	FS	GS
Коди команд	+	–	–	–	–	–
Дані індексовані за допомогою регістра BP, або EBP	S	S	+	S	S	S
Елементарні дані, та індексовані дані, крім даних де зміщення формується з врахуванням регістра BP, або EBP	S	+	S	S	S	S
Стек	–	–	+	–	–	–
Стрічка–джерело	S	+	S	S	S	S
Стрічка–приймач	-	-	-	+	S	S
Таблиця	S	+	S	S	S	S

У таблиці використані позначення:

– - сегмент по замовчуванню;

S - сегмент, може використовуватись при наявності спеціальної однобайтової команди (префіксу перекриття сегменту);

– - сегмент неможливо використовувати.

Формування зміщення залежить від режиму адресації, типу команди та типу передачі по шині (вибірка команди, вибірка даних і т.п.). При зчитуванні коду команди зміщення береться з регістра IP (EIP). При роботі зі стеком

зміщення береться із регістра SP (ESP). При роботі з "простими" даними формування зміщення здійснюється в залежності від типу команди, що виконується. У МП і386 зняті обмеження на використання регістрів у формуванні зміщення у 32 бітному режимі адресації. Загалом ефективне зміщення Offset в команді вираховується за формулою:

$$\text{Offset} = (\text{Base reg}) + (\text{Index reg}) * (\text{Scale}) + (\text{Displacement}),$$

У формулі використані такі позначення полів: Base reg – базовий регістр, Index reg – індексний регістр, Scale – масштаб, Displacement – зміщення у команді. Залежно від типу адресації даних можливі значення полів змінюються. Можливі значення полів наведені у таблиці 16.

Таблиця 16. Можливі значення полів у формулі формування зміщення

Поле	16-розрядна адресація	32-розрядна адресація
Base reg	BX або BP	Будь-який 32-розрядний регістр загального призначення
Index reg	DI або SI	Будь-який 32-розрядний регістр загального призначення, крім ESP
Scale	1	1, 2, 4, 8
Displacement	0, 8 або 16 біт	0, 8, 16 або 32 біти

При табличній адресації адреса формується шляхом сумування вмісту регістра AL з регістром BX. Адресація використовується при виконанні команди перекодування по таблиці XLAT.

У випадку використання строкових команд, що використовують строку як джерело даних у якості зміщення обов'язково використовується вміст регістра SI (ESI). Якщо ж команди використовують строку як приймач даних, у якості зміщення використовується вміст регістра DI (EDI).

Принцип трансляції логічної адреси у фізичну у мікропроцесорі і386 проілюстрований на рис. 33. Уміст селектора використовується для пошуку у локальній чи глобальній дескрипторній таблиці запису (дескриптора), що визначає властивості сегмента та його базову адресу. Базова адреса додається до зміщення у результаті отримується лінійна адреса. Одночасно з формуванням адреси перевіряється доступність сегмента та права доступу до нього.

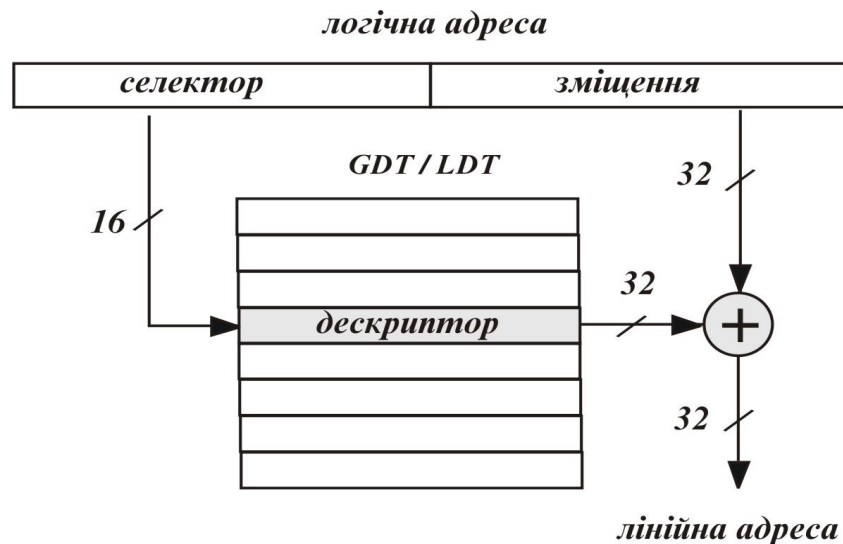


Рис. 33. Формування лінійної адреси в захищеному режимі

7.6. Сторінки

У МП i386 з'явилась можливість використовувати сторінкову адресацію. Сторінки мають фіксований розмір, вони не пристосовані під програмні структури, але значно полегшують роботу операційної системи з віртуальною пам'яттю.

Операційна система може розділити лінійний адресний простір на сторінки незалежно від розбиття логічного адресного простору задачі на сегменти. З погляду операційної системи сторінки є зручними елементами для розподілу й перерозподілу пам'яті, оскільки вони усі мають однаковий розмір. Сторінки також є засобом захисту частин великих сегментів і, що особливо важливо, забезпечують зручний спосіб організації віртуальної пам'яті.

Сторінка в i386 має довжину 4 кбайт. Цей розмір до збільшує продуктивність з двох причин:

- ◇ по-перше, він забезпечує оптимальне співвідношення розміру сторінок з розміром кеш-пам'яті;
- ◇ по-друге, розмір у 4 кбайти є зручним для дискових передач.

Операційна система включає сторінкову переадресацію у i386 встановлюючи біта PG у регістрі CR0. Коли сторінковий механізм включений, мікропроцесор транслює лінійну адресу у фізичну адресу, використовуючи сторінкові таблиці.

Сторінкові таблиці із 386 організовані за принципом дворівневої ієрархії. Кожна задача може мати свій каталог таблиць сторінок (PDE). Кожний запис каталога сторінок указує на свою таблицю сторінок (PTE), що містить дані для формування фізичної адреси комірки пам'яті.

Системний реєстр CR3 мікропроцесора i386 указує на фізичку адресу початку каталогу таблиць сторінок виконуваної задачі. Мікропроцесор оновлює вміст реєстра CR3 при кожному перемиканні задачі, зчитуючи нове значення адреси початку каталогу сторінок з сегмента стану нової задачі (TSS).

Формат регістра CR3 зображений на рис. 34. У 24 старших розрядах регістра зберігається фізична адреса початку каталогу сторінок (PDE Addr). Молодші 12 розрядів мають мати нульове значення.

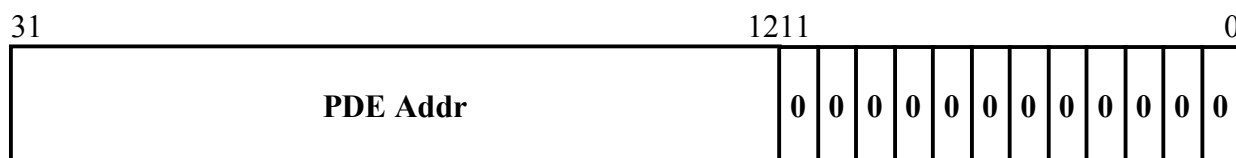


Рис. 34. Формат CR3

Каталог таблиць сторінок (PDE) має довжину 4 кбайти. Він містить посилання на 1024 таблиці сторінок (PTE). Кожен елемент таблиці сторінок описує 4 Мбайти адресного простору.

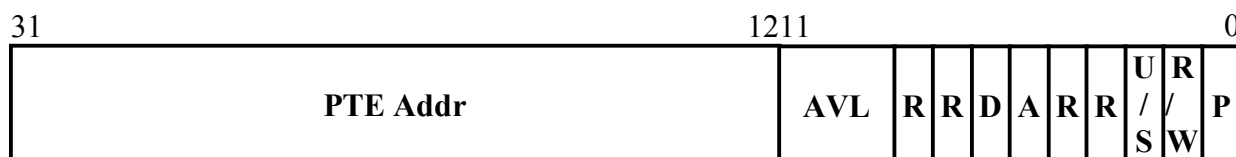


Рис. 35. Формат елемента каталогу таблиць сторінок

Каталог таблиць сторінок за своєю організацією аналогічний таблиці дескрипторів, за винятком того, що кожен елемент каталогу таблиць сторінок таблиці містить фізичну базову адресу сторінки розміром 4 кбайт. Формат елемента каталогу таблиць сторінок зображений на рис. 35. Поле PTE Addr задає фізичну адресу початку таблиці сторінок. Таблиця сторінок вирівнюється, щоб молодші 12 розрядів адреси були рівні 0. Молодші 12 розрядів елемента каталогу таблиць сторінок містять атрибути таблиці сторінок.

Поле D визначає необхідність запису зміненої сторінки. Встановлюється

мікропроцесором в "1", якщо був виконаний запис у таблицю сторінок або сторінку.

Поле А встановлюється в "1" мікропроцесором у процесі виконання операцій зчитування чи запису в сторінку. Указує, що до сторінки відбувався доступ.

Поле U/S використовується для обмеження доступу до сторінок операційної системи (сторінок супервізора) з боку програм користувача. Значення "0" відповідає сторінкам супервізора, "1"- сторінкам програми користувача. Програмами користувача вважають будь-які програми, виконувані на рівні привілеїв 3, усі решта рівнів привілеїв вважаються рівнями супервізора.

Біт W/R використовується для дозволу запису у сторінку. Якщо біт встановлено в 1, то дозволено запис у всі сторінки. Біт використовується для організації захисту від записування даних на рівні.

Біт Р визначає присутність у пам'яті. Встановлюється в 1, якщо таблиця сторінок, яка визначається елементом каталогу, розміщена у оперативній пам'яті. Цей біт використовується для організації віртуальної пам'яті на рівні сторінок.

Біти AVL доступні для використання операційною системою. Операційна система може писати у ці біти свою інформацію, котра визначає властивості таблиці сторінок. Біти позначені знаком R зарезервовані і встановлюються в 0 для узгодження з подальшими моделями процесорів. Не всім елементам каталогу таблиць сторінок обов'язково мають відповідати реальні таблиці. Частина таблиці сторінок може знаходитись на диску або узагалі бути відсутня. Для таких елементів біт Р має бути скинутий у нуль.

Кожна таблиця сторінок теж має розмір 4 кбайти та складається із елементів формат котрих зображено на рис. 36.

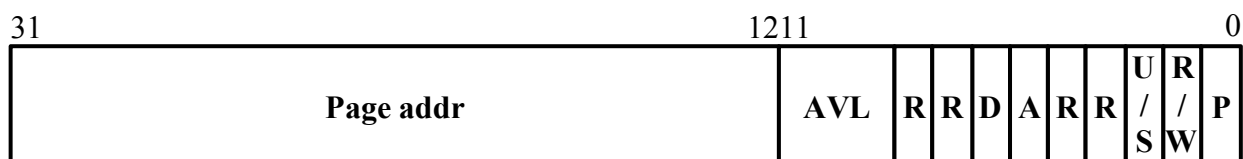


Рис. 36. Елемент таблиці сторінок

Поле Page Addr зберігає старші розряди фізичної адреси сторінки у пам'яті. Молодші розряди фізичної адреси початку сторінки завжди рівні 0. Молодші розряди таблиці зберігають атрибути сторінки. Біти у атрибутах сторінки мають таке ж значення як і у атрибутах елемента каталогу сторінок, проте діють вони лише на сторінку.

Фізична адреса на основі логічної формується за схемою зображеною на рис. 37.

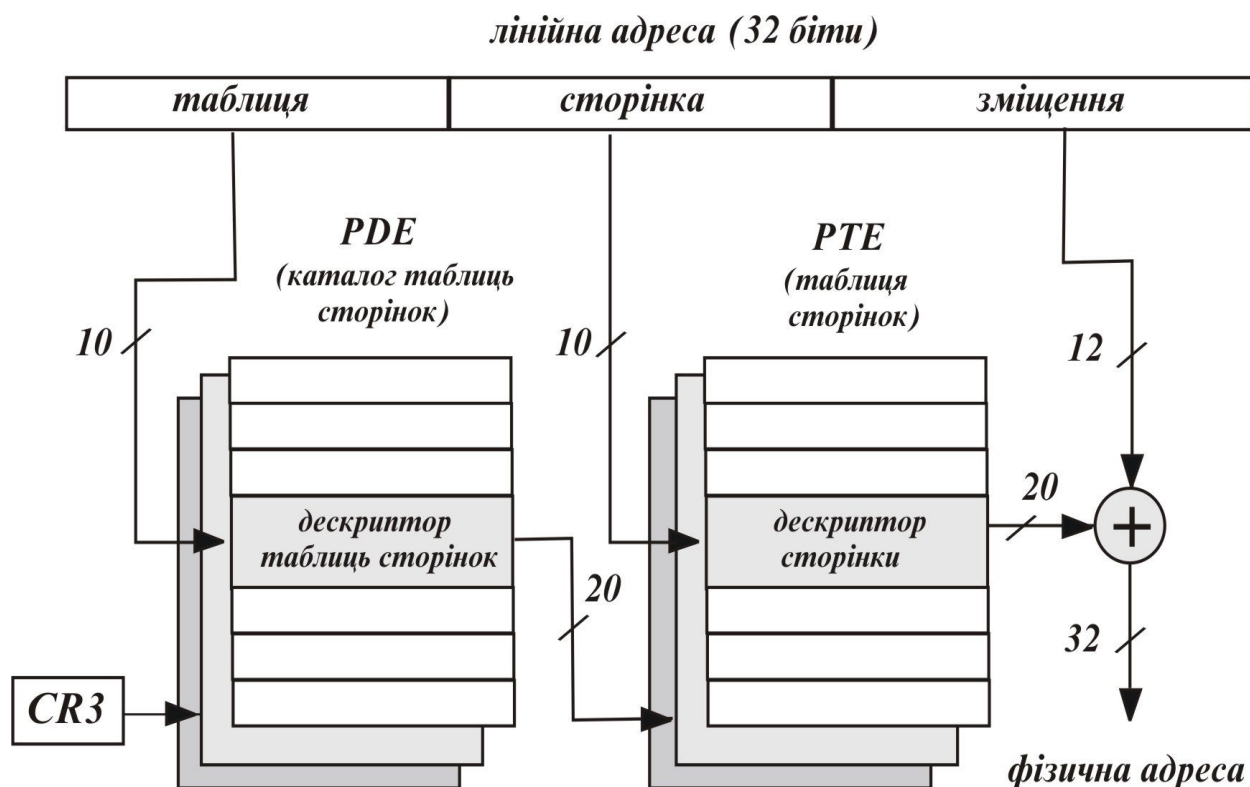


Рис. 37. Формування сторінкової адреси в захищеному режимі.

10 старших розрядів лінійної адреси адресують один зі каталогів таблиць PDE, розташування яких у пам'яті визначається вмістом 32-розрядного системного регістра CR3. 20 старших його розрядів містять базову адресу каталогу таблиць сторінок, а 12 молодших не визначені. Використовуючи старших 10 розрядів лінійної адреси МП здійснює пошук елемента у каталозі таблиць сторінок, який задає базову адресу таблиці сторінок та перевіряє права доступу.

Настуні 10 розрядів лінійної адреси визначають елемент таблиці сторінок. Таблиця сторінок містить 1024 дескриптори, які визначають фізичну

адресу початку сторінки пам'яті розміром 4 кбайти. Молодші 12 розрядів визначають зміщення у середині сторінки. Разом з 20 розрядами елемента таблиці сторінок вони формують фізичну адресу комірки пам'яті.

Механізм трансляції сторінок використовується мікропроцесором, якщо старший біт системного регістра CR0 (PG) встановлено в 1. Якщо ж біт рівний "0", то фізична адреса рівна лінійній, а регістр CR3, каталог таблиць сторінок, та таблиці сторінок у формуванні фізичної адреси участі не приймають.

Операційна система, що працює на МП i386 може використовувати розподіл адресного простору за допомогою сегментів та сторінок. Оскільки сегменти можуть перекривати весь адресний простір МП, операційна система може дати задачі лінійний адресний простір без сторінок (як у процесорі i8080), або лінійний адресний простір зі сторінок, або адресний простір із сегментів (як у процесорі i8086), або адресний простір із сегментів з поділом сегментів на сторінки. Багатоступінчаста трансляція адреси в i386 виконується швидко, типовий час обчислення зміщення і трансляції логічної адреси у фізичну – 1,5 такти.

Для того, зменшити час на пошук адреси у сторінкових таблицях, i386 розміщає довідкову інформацію про 32 останні сторінки, що ним використовувалися, у спеціальний кеш у середині МП. Цю пам'ять називають буфером трансляції сторінок (TLB). Тільки у випадку, якщо мікропроцесор не знаходить інформації про сторінку в TLB, він звертається до сторінкових таблиць, які розміщуються у пам'яті. Як правило, 98-99% звертань потрапляють у TLB, не вимагаючи звертань до основної пам'яті для трансляції. Коли необхідної інформації у TLB не має, мікропроцесор замінює один з елементів TLB новим елементом, який буде використовуватися надалі.

8. Захист

Процесор i386 забезпечує кілька механізмів захисту. Одна з форм захисту – відділення просторів адрес різних задач за допомогою таблиць дескрипторів сегментів і сторінкових таблиць. Додатково до відділення однієї задачі від

іншої i386 забезпечує можливість захисту операційної системи від прикладних задач, захист однієї частини операційної системи від інших її частин і захист задач від деяких власних помилок. Вимоги до захисту найповніше можуть бути задоволені за допомогою набору вибірково використовуваних функцій захисту, уведених у i386:

- ◇ поділ адресних просторів задач;
- ◇ введення 4 рівнів привілеїв ;
- ◇ використання привілейованих команд
- ◇ поділ сегментів за типами (наприклад, кодовий сегмент чи сегмент даних);
- ◇ введення прав доступу до сегментів і сторінок (наприклад, право тільки читання чи тільки виконання);
- ◇ перевірка меж сегментів.

Ступінь і тип використовуваної системи захисту залежить від конкретного застосування. Прості системи можуть не використовувати захист.

Система захисту i386 також дозволяє спростити процес відлагодження програм, використовуючи переривання за помилками для конкретних задач.

Для збереження максимальної продуктивності всі перевірки захисту в i80386 виконуються у момент виконання команди. Усі властивості захисту мікропроцесора i386 реалізовані усередині кристала, тому дії з перевіряння захисту не впливають на продуктивність мікропроцесора.

8.1. Привілеї

Більшість властивостей захисту мікропроцесора i386 базуються на ієрархії привілеїв. У кожному дескрипторі сегмента є поле, що визначає рівень привілею пов'язаного з ним сегмента, поле може набирати значення від 0 до 3. Нульове значення поля указує на найбільш привілейований рівень, а 3 – найменш привілейований.

Система без захисту може бути реалізована простим віднесенням усіх процедур у сегмент (чи сегменти), чий рівень привілею дорівнює 0.

Традиційний поділ на супервізора і користувача може бути реалізований віднесенням прикладної задачі у сегмент із рівнем привілею 3, а процедур супервізора – у сегмент, де рівень привілеїв 0. Операційна система може також використовувати рівні привілеїв 1 і 2, якщо це необхідно.

Наприклад, найбільш критичні процедури операційної системи (ядро) можуть мати рівень привілеїв 0. У такому випадку ядро операційної системи захищене від усіх інших програм.

Рівень привілеїв 1 може бути використаний для функцій операційної системи, що є менш критичними і частіше змінюються чи розширюються, наприклад, для драйверів пристроїв.

Рівень 2 можуть використовувати розробники оригінального обладнання. У цьому випадку програми розробників оригінального устаткування захищені від програм кінцевих користувачів.

Рівень привілеїв задачі визначає підмножину сегментів та сторінок до котрих програма може звертатися, а також команди, що використовуються у програмі. Будь-яка спроба задачі використовувати більш привілейований сегмент чи сторінку призводить до зупинки роботи програми та виникнення особливої ситуації захисту.

8.2. Використання привілейованих команд

Рівень привілеїв задачі визначає команди, які можуть бути використані. Мікропроцесор i386 має підмножину команд, виконання яких має бути ретельно проаналізоване, щоб запобігти серйозним руйнуванням системи. Усі команди, що завантажують нові значення у системні регістри, є прикладами привілейованих команд. Тільки задача, яка працює на рівні привілеїв 0, може виконувати привілейовані команди.

8.3. Ввід-вивід

Системи, що базуються на i386, можуть визначати пристрої вводу-виводу в просторі пам'яті процесора чи в окремий простір вводу-виводу. До пристроїв вводу-виводу у просторі пам'яті можна звертатися для читання чи запису,

використовуючи такі команди звертання, як MOVE, OR чи аналогічні. Пристрої, у пам'яті, можуть бути захищені за допомогою стандартного механізму захисту сегмента і сторінки процесора i386.

Крім адресного простору пам'яті, мікропроцесор i386 має 64 кбайтовий адресний простір вводу-виводу. До пристроїв у цьому просторі звертаються за допомогою команд вводу IN, виводу OUT, вводу рядка INS і виводу рядка OUTS. Перші дві команди передають байт, слово чи подвійне слово через регістр AL, AX чи EAX залежно від формату команди. Останні дві передають рядок байтів, слів чи подвійних слів у пам'ять чи з пам'яті.

Виконання команд вводу-виводу i386 залежать від рівня привілей та сегмента стану задачі. У регістрі ознак є поле, що називається рівнем привілеїв вводу-виводу (IOPL), яке визначає мінімальний рівень привілеїв, на якому працююча задача може виконувати команди вводу-виводу. Наприклад, якщо IOPL задачі дорівнює 1, то задача не може видавати команди вводу-виводу, крім випадків, коли вона працює на рівні привілей 0 чи 1, або робота програми з цим рівнем привілеїв не дозволена у бітовій карті вводу-виводу.

8.4. Захист на рівні сегментів

Дескриптори в регістрах LDT і GDT визначають логічний адресний простір задачі. Сегменти, визначені у цих таблицях, можуть бути доступні, оскільки таблиці дескрипторів володіють інформацією, необхідною для обчислення адреси сегмента. Однак адресований сегмент не може бути доступним для деяких операцій через додаткову перевірку захисту, яку здійснює мікропроцесор. МП перевіряє кожне звертання до сегмента для узгодження з атрибутами захисту сегмента.

Щоб отримати доступ до сегмента, програмі слід мати принаймні такий же привілей, який є у сегмента. Наприклад, програма, що працює на рівні 3, може звертатись тільки до тих сегментів, у яких рівень привілеїв також дорівнює 3, у той час, як програма, що працює на рівні 0, може звертатися до усіх сегментів у своєму адресному просторі.

Адресовані у сегменті дані повинні знаходитися усередині сегмента. Межі сегмента дозволяють мікропроцесору знаходити програмні помилки: переповнення стека, невірні покажчики й індекси масивів, а також неправильні адреси викликів та переходів.

У випадках, коли операційна система може визначити, що звертання за межі сегмента не є помилкою (переповнення стека є прикладом для деяких систем), операційна система може розширити сегмент (наприклад, додаванням сторінок до нього) і почати виконання команди з початку.

Кожен дескриптор містить поле типу. Мікропроцесор перевіряє на відповідність команді типу сегмента. Несистемні сегменти поділяються на сегменти кода та даних, що дозволяють, наприклад, процесору знайти спробу запису в існуючий код.

Системні дескриптори також мають визначений тип, що дозволяє процесору перевірити відповідність типу операції властивостям сегмента. Дескриптор сегмента може мати права, що обмежують операції, які можна здійснювати над пов'язаним із ним сегментом. Є можливість заблокувати усі операції над сегментом коду, крім операцій зчитування коду команди. Сегменти даних можуть бути указані або як доступні тільки для читання, або як доступні для читання і запису.

У випадку необхідності, одна задача може мати доступ до сегмента у режимах читання та записування даних, а інша – лише у режимі читання, або виконання коду.

8.5. Захист на рівні сторінок

Системи, можуть також використовувати захист сторінок. Захист сторінок можна також застосовувати до окремих частин великих сегментів. Аналогічно дескриптору сегмента елемент таблиць сторінок має набір атрибутів захисту, і мікропроцесор i386 перевіряє кожне звертання до сторінки на відповідність цим атрибутам.

Елемент сторінкової таблиці може бути відзначений одним із двох рівнів

привілеїв: користувальницьким чи супервізорним.

Користувальницький рівень відповідає рівню привілеїв 3, а супервізорні сторінки можуть бути доступні тільки задачам, які працюють з рівнями привілеїв 0, 1 чи 2. Користувальницька сторінка може бути відзначена як доступна тільки для читання чи для читання і запису.

Процесор i386 перевіряє атрибути захисту сторінок після того, як упевнився, що метод доступу відповідає атрибутам сегмента. Таким чином, захист сторінок є зручним засобом для операційної системи реалізувати додатковий захист частин сегментів.

8.6. Захист при передачі керування. Шлюзи. Міжсегментні переходи

Для міжсегментного передавання керування використовуються команди JMP, CALL, INT, RET, IRET. У командах JMP та CALL указується адреса, куди буде відбуватись перехід у форматі селектор:зміщення. У захищеному режимі передавання керування можливе як пряму на сегменти коду, так і через шлюзи.

Передача керування з використанням селекторів сегментів коду можливе лише на сегмент, котрий має рівень привілеїв DPL, як і CPL задачі (якщо у полі доступу сегмента з відповідним селектором біт C рівний 0), або на сегмент у котрого $CPL \leq DPL$ (якщо у полі доступу сегмента з відповідним селектором біт C рівний 1). У будь-якому випадку рівень привілеїв програми не зміниться. Передача керування на сегменти коду з рівнем DPL, котрий не підпадає під указані умови не відбувається.

Перехід до виконання коду високим рівнем привілеїв можливий з використанням шлюзів (Gate), або при переключенні задач. Для забезпечення переключення задачі необхідно, щоб селектор у команді переходу вказував на сегмент стану іншої задачі, або на шлюз задачі.

Існують декілька типів шлюзів: шлюз виклику підпрограми або міжсегментного переходу (Call Gate), шлюз задачі (Task Gate), шлюз

переривання (Interrupt Gate) шлюз пастки (Trap Gate).

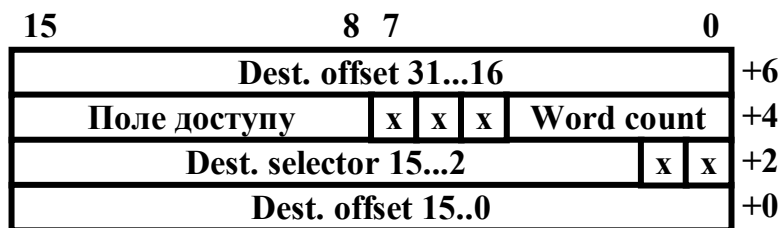


Рис. 38. Дескриптор шлюзів

Формат селектора шлюзу співпадає з форматом селектора сегмента кода. Він представляє собою поєднання селектора котрий поміщається у команді переходу чи виклику у позицію сегмента та дескриптора, котрий визначається локальною або глобальною дескрипторною таблицею за загальними правилами.

Можливе використання шлюзів у форматі i80286 та у форматі i386. Тип шлюза та формат визначається полем TYPE у полі доступу дескриптора. Формат шлюзів у стилі процесора i80286 був описаний у попередньому розділі, формат шлюзів у стилі i386 зображений на рис. 38.

На рисунку використані такі позначення: Dest. offset – зміщення початку процедури відносно початку сегмента, Dest. selector – селектор сегмента де розміщається код котрий викликається. Поле Dest. selector для шлюзів виклику, переривань та пасток задає селектор відповідного сегмента коду, а для шлюзів задач – селектор сегмента TSS. Поле доступу містить код прав доступу до нового сегменту коду, а Word count містить кількість байт які необхідно скопіювати у стек викликаної задачі.

Не усі шлюзи можуть використовуватись у всіх дескрипторних таблицях. Шлюз виклику не можна використовувати у IDT, а шлюзи переривань та пасток може використовуватись лише у IDT.

Не усі поля використовуються у шлюзах різного типу. Поле Word count використовується лише у шлюзах виклику, а поле Dest. offset не використовується в шлюзах задач.

Шлюз задачі може використовуватись у будь-якій з дескрипторних таблиць. Його особливість – у якості сегмента указаний сегмент TSS. Поле зміщення не використовується та може мати будь-яке значення.

Особливістю шлюза переривань є скид ознаки дозволу переривань IF при вході у процедуру, що викликається. Попереднє значення регістра ознак зберігається у стеку, тому при виході із процедури обробки ознака IF приймає попереднє значення.

Шлюз пастки, працює аналогічно шлюзу переривань, але замість ознаки IF скидає ознаку TF.

Алгоритм переходів з використанням шлюзів:

- ◇ Якщо рівень привілей викликаного сегмента, чисельно менше CPL, відбувається зміна стека, із відповідних полів TSS завантажуються нові значення SS, (ESP) SP;
- ◇ Якщо була зміна стека, то в новий стек зберігається попереднє значення SS, (ESP) SP ;
- ◇ Якщо була зміна стека, а шлюз є шлюзом виклику, то в новий стек копіюються Word Count слів зі стеку попередньої програми;
- ◇ Якщо відбувається перехід на шлюз переривання чи пастки зберігається значення регістра EFlags (Flags);
- ◇ Зберігається біжуче значення CS, (E)IP;

Із дескриптора шлюзу у регістри CS:EIP завантажується нова адреса, задана відповідними полями дескриптора. Розрядність регістрів (E)SP та (E)IP визначається розрядністю адрес та операндів, котра використовується за замовчуванням.

8.7. Режим віртуального 86

Процесор i386 може виконати програми i8086 в одному з 2 режимів: реальному або режимі віртуального 86. Режим віртуального 86 емулює середовище i8086 усередині захищеного багатозадачного середовища i386. У режим віртуального 86 процесор користується усіма можливими методами захисту та можливістю роботи у багатозадачному середовищі. Задачі, що виконуються можуть бути переведені у режим віртуального 86 незалежно одна від одної.

Коли процесор працює у режимі віртуального 86, то поводить ся, як і мікропроцесор i8086, але після перемикання до "нормальної" задачі, процесор працює як 16- чи 32-бітний процесор у захищеному режимі. Основна задача режиму віртуального 86 – дозволити виконання програм реального режиму у захищеному режимі.

Вмикання режиму віртуального 86 забезпечує ознака VM86 у регістрі ознак, що завантажується з TSS задачі.

У режимі віртуального 86 кожній задачі надається адресний простір в 1 Мбайт. Лінійна адреса операндів формується за правилами реального режиму. Вміст сегментного регістра множиться на 16 та додається до зміщення. Щоб задачі, що працюють у режимі віртуального 86, не перетинатися одна з одною, операційна система може використовувати сторінковий механізм i386 для переміщення лінійних адресних просторів задач, що працюють у режимі віртуального 86, у різні області фізичного адресного простору.

Використання сторінкового механізму у режимі віртуального 86 дозволяє операційній системі також використовувати віртуальну пам'ять для задач віртуального 86 процесора так само як і для задач i80386.

9. Віртуальна пам'ять

Основна ідея віртуальної пам'яті полягає у тому, щоб зберігати (і оновлювати) вміст великої віртуальної пам'яті на диску, "підкачуючи" окремі ділянки віртуальної пам'яті в реальну оперативну пам'ять.

Можна, наприклад, зберігати усі використовувані програмою сегменти на диску, а у фізичну пам'ять записувати тільки сегменти, необхідні для виконання програми в даний момент.

Віртуальна пам'ять дозволяє використовувати програми, що потребують максимальний розмір пам'яті більший ніж об'єм фізичної пам'яті (ОЗП) у системі. Віртуальну пам'ять реалізує операційна система з відповідною апаратною підтримкою. Мікропроцесор i386 забезпечує роботу із системами віртуальної пам'яті із сегментною чи сторінковою організацією.

Системи з віртуальною пам'яттю можуть ґрунтуватися або на сегментах, або на сторінках. В обох випадках основна ідея віртуальної пам'яті полягає у тому, щоб використовувати значно дешевшу дискову пам'ять замість напівпровідникової. Операційна система у випадку з віртуальною пам'яттю запам'ятовує усі сегменти чи сторінки у великій області дискової пам'яті, яку часто називають "областю обміну".

Значно менша фізична (реальна) пам'ять містить тільки найчастіше використовувані сегменти або сторінки. Доти, поки сегменти чи сторінки, запам'ятовані на диску, не використовуються, система з віртуальною пам'яттю буде такою ж швидкою, як і система зі значно більшою оперативною пам'яттю. Для ефективного використання механізму віртуальної пам'яті необхідно:

- ◇ наявність біта для кожного сегмента чи сторінки, який повідомляє мікропроцесору (чи пристрою керування пам'яттю), чи є даний сегмент чи сторінка в пам'яті, чи необхідно завантажити його (її) з диска;
- ◇ механізм особливого переривання, за допомогою якого мікропроцесор може вказати операційній системі про необхідність завантаження у пам'ять відсутнього сегмента чи сторінки;
- ◇ команди перезавантаження, що дозволяють мікропроцесору вибирати заново команду після того, як операційна система завантажила тільки що відсутню сторінку у пам'ять і помітила її присутність.

Мікропроцесор i386 має усі ці необхідні властивості. Як дескриптори, так і елементи сторінкових таблиць мають біт присутності, тому можуть бути використані як основа для побудови віртуальної пам'яті. Обмін сегментами між пам'яттю і диском є виправданим, коли сегменти відносно малі. Коли сегменти можуть бути великі, обмін сторінками ефективніший завдяки фіксованому їх розміру.

9.1. Віртуальна пам'ять на рівні сегментів

Сегментна віртуальна пам'ять більше підходить для невеликих 16-бітних

систем, в яких довжина сегмента не перевищує 64 кбайти. МП і386 забезпечує роботу із сегментами об'ємом до 4 Гбайт; тому в більшості систем на базі і386 системи віртуальної пам'яті використовують сторінкову організацію. Проте віртуальна пам'ять на рівні сегментів дозволяє створити більш надійний захист програм одна від одної та від операційної системи.

При використанні віртуальної пам'яті непотрібні у даний момент сегменти записуються на жорсткий диск, а звільнене місце використовується потрібними даними.

Операційна система веде облік сегментів, що знаходяться в пам'яті та на диску. Два поля дескриптора в полі доступу: біт присутності сегмента в пам'яті P і біт звертання до сегмента пам'яті A призначені для апаратної реалізації обліку сегментів. У дескрипторі таких "непотрібних" сегментів, у полі $Base$ указується розміщення сегмента на жорсткому диску, одночасно у полі доступу біт P встановлюється в 1. Коли відбувається звертання до даних розміщених у сегменті з бітом $P = 1$ у полі доступу, викликається спеціальне виключення ($\#NP$), процедура обробки котрого зчитує потрібні дані з пам'яті, змінює дескрипторні таблицю, та повторно виконує команду, котра звернулась до вказаних даних.

9.2. Віртуальна пам'ять на рівні сторінок

Сторінки мають фіксований розмір, що полегшує оперування ними. Для кожної сторінки МП має біт присутності P , а також реєструє звертання до сторінки.

Всі біти елементів таблиці сторінок, котрі відсутні у пам'яті, за винятком біта присутності P визначаються операційною системою. Вони надають операційній системі зручне місце для запам'ятовування дискової адреси відсутньої сторінки.

У загальному операційна система зі сторінковою віртуальною пам'яттю пересилає відсутні сторінки з диска у момент звертання до відсутньої сторінки. Коли в процесі трансляції логічної адреси мікропроцесор отримує лінійну адре-

су, що посилається на елемент сторінкової таблиці, чий біт присутності Р скинутий, у мікропроцесорі виникає особлива ситуація "відмова сторінки" (#PF). При вході у процедуру обробки в відмови сторінки регістр CR2 містить лінійну адресу, пов'язану з відсутньою сторінкою. За цією адресою процедура обробки відмови сторінки може знайти відповідний йому елемент сторінкової таблиці, транслюючи лінійну адресу. Визначивши дискову адресу відсутньої сторінки, МП за допомогою операційної системи знаходить вільне місце у пам'яті та передає сторінку з диска. МП змінює адресне поле в елементі сторінкової таблиці та біт присутності. Після цього мікропроцесор заново виконує команду, у якій відбулася сторінкова помилка.

Крім завантаження сторінок за запитом, операційна система повинна мати набір вільних сторінок, що можуть бути розподілені при опрацюванні відмов сторінок. Для збільшення набору вільних сторінок операційна система має знати, яку сторінку слід звільнити.

Перш ніж звільнити кадр, операційна система повинна записати цю сторінку на диск, якщо сторінка була модифікована з моменту її завантаження. Для цього архітектура i386 має біт доступу А і біт зміни D в кожному елементі сторінкової таблиці. Мікропроцесор змінює ці біти автоматично, і для всіх наявних у пам'яті сторінок встановлює біт доступу завжди, коли відбувається читання чи записування, та встановлює біт зміни, коли відбувалося записування даних у сторінку. Періодично переглядаючи і скидаючи бітів доступу, операційна система може визначити сторінки, які давно не використовували. Коли операційна система вибрала сторінку що давно не використовувалась, сторінка буде записана на диск, якщо мікропроцесор встановив її біт зміни.

Висока продуктивність у роботі з віртуальною пам'яттю забезпечується в i386 використанням внутрішньої кеш-пам'яті для збереження сторінкової інформації. Ця кеш-пам'ять (TLB) зберігає інформацію про розподіл адрес 32 сторінок, що використовувались. Сторінки віртуальної пам'яті i386 мають обсяг 4 кбайти. Зберігаючи одночасно розподіл 128 кбайт пам'яті, буфер TLB дозволяє мікропроцесору перетворити адреси всередині кристала, не

звертаючись до таблиці сторінок, яка зберігається у пам'яті. У типових системах 98-99% пошуку адрес буде здійснюватись через буфер TLB.

10. Перемикання задач

Архітектура i386 своїми забезпечує роботу багатозадачних операційних систем. У багатозадачній системі кожен вид діяльності, що може здійснюватися одночасно з іншими, називається задачею ("процесом"). Програми, звичайно складені так, ніби вони виконувалися б на окремих мікропроцесорах із загальною пам'яттю.

Багатозадачна операційна система емулює декілька мікропроцесорів, надаючи кожній задачі "віртуальний мікропроцесор". У кожен момент часу операційна система надає реальний мікропроцесор лише одній із задач. Для забезпечення враження, що кожна задача має свій МП, операційна система через певний час переключає реальний мікропроцесор між задачами. У системній архітектурі i386 для операції зміни задачі передбачені сегменти стану задачі.

10.1. Регістр TR

Для організації багатозадачної роботи мікропроцесора в захищеному режимі використовують регістр задачі TR, який містить селектор сегмента стану задачі TSS. Сегмент стану TSS зберігає копії усіх регістрів МП, які використовує мікропроцесор. Формат регістра TR зображений на рис. 39.

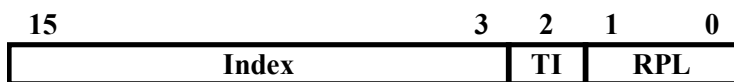


Рис. 39. Формат регістра TR

Два молодші розряди селектора (RPL), визначають чотири рівні привілеїв, що запитує програма при доступі до сегмента. Наступний біт селектора (TI) указує тип дескрипторної таблиці. Якщо біт TI рівний 0 вибирається глобальна дескрипторна таблиця, якщо 1 – локальна дескрипторна таблиця. Рекомендовано для сегмента стану задачі користуватись лише глобальною дескрипторною таблицею. Старші тринадцять розрядів селектора (Index), вказують на номер 64-розрядного запису з інформацією про базову

адресу, розмір сегмента та права доступу.

З регістром TR зв'язаний схований регістр, що містить інформацію про права доступу, розмір та базову адресу сегмента стану задачі. У момент перемикання задач у нього записуються дані із дескрипторної таблиці.

10.2. Сегмент стану задачі

Сегмент стана задачі (TSS) є однією із кількох структур даних, зумовлених системною архітектурою i386. У сегменті TSS зберігається стан віртуального процесора задачі. Кожна задача i386 представлена своїм TSS, що складається з двох частин. Молодша частина TSS визначена системною архітектурою i386 і містить значення усіх регістрів мікропроцесора. Старша частина TSS може бути визначена операційною системою для збереження даних, пов'язаних із конкретною задачею, наприклад, пріоритету виконання, дескрипторів файлів і т.д. Для створення своєї задачі операційна система формує TSS та заповнює його значеннями, необхідними задачі для початку її виконання.

МП i386 може працювати з сегментами стану задачі у стилі i80286 та у стилі i386. Визначення типу сегмента стана задачі здійснюється за допомогою поля Type у полі Access дескриптора сегмента. Формат сегментів стану задачі у стилі i80286 був розглянутий при описі відповідного МП. Формат сегмента задачі TSS зображено на рис. 40.

Сегмент TSS зберігає копії 16-бітних сегментних регістрів, вміст регістра локальної таблиці дескриптора LDTR, вміст 32-розрядних регістрів загального призначення, регістра ознак EFlags, лічильника команд EIP, вміст системного регістра CR3, а також вказівники стека для трьох пріоритетних рівнів (поля SS2:ESP2, SS1:ESP1, SS0:ESP0).

Поле LINK використовується для посилання на сегмент TSS при вкладеному виклику задачі. Біт T призначений для відлагодження ("1").

Бітова карта I/O Permission Map визначає порти, які дана задача може використовувати. Кожен розряд карти вказує на певний порт вводу-виводу. Встановлення відповідного розряду карти в 1 забороняє доступ до порта,

встановлення в 0 дозволяє його.

31		16		15		0		
00h		Back Link						00
ESP0								04
00h		SS0						08
ESP1								0C
00h		SS1						10
ESP2								14
00h		SS2						18
CR3								1C
EIP								20
EFLAG								24
EAX								28
ECX								2C
EDX								30
EBX								34
ESP								38
EBP								3C
ESI								40
EDI								44
00h		ES						48
00h		CS						4C
00h		ES						50
00h		CS						54
00h		ES						58
00h		CS						5C
00h		LDT						60
Bit Map Offset		00h				T	64	
Available								
31	24	23	16	15	8	7	0	Bit Map Offset
63	56	55	48	47	40	39	32	Bit Map Offset+04h
95	88	87	80	79	72	71	64	Bit Map Offset+08h
127	120	119	112	111	104	103	96	Bit Map Offset+0Ch
...								
I/O Permission Bitmap								
65535	65528	65527	65520	65519	65512	65511	65504	Bit Map Offset+1FFCh
						FFh		Bit Map Offset+2000h

Рис. 40. Формат стана задачі i386

Останнім байтом карти вводу-виводу має бути байт, що складається з одних одиниць. Бітова карта I/O Permission Map може бути скорочена шляхом зменшення довжини сегмента TSS, але у будь-якому випадку її останній байт

має бути рівний 0FFh. Положення початку бітової карти визначається словом Bit Map Offset, котре задає зміщення бітової карти відносно початку сегмента.

Поле Available може використовуватись операційною системою з будь-якою метою. У випадку необхідності аналогічне поле може розташовуватись також після бітової карти вводу-виводу.

10.3. Зміна задачі

Мікропроцесор i386 зберігає селектор і дескриптор поточної задачі у своєму регістрі TR. Коли відбувається переключення на нову задачу МП зберігає у сегменті TSS значення регістрів, завантажує у TR селектор нової задачі. На основі селектора за допомогою GDT визначається дескриптор нової задачі. Одержавши лінійну адресу нового TSS, процесор завантажує свої регістри значеннями із нового TSS. У поле Back Link TSS нової задачі зберігається значення селектора старої задачі, якщо нова задача є вкладеною. Виконання програми продовжується з команди, на яку вказує лічильник команд нової задачі.

Для поновлення старої задачі операційна система повинна видати команду переходу (чи повернення для вкладки задачі), після цього виконання старої задачі продовжується із команди, що є наступною після команди переходу на задачу, що призупинила її виконання. Процес зміни задач займає коло 17 мкс при робочій частоті 16 МГц й за відсутності тактів очікування.

11. Система команд i386

11.1. Основні типи даних МП

Базові типи даних МП i386: біти, байти, 16-бітні слова та 32-бітні подвійні слова. Найменша інформаційна одиниця – біт використовується у командах, що встановлюють та скидають біти регістра ознак, а також команди сканування та пошуку бітів.

Більш широко використовуються байти.

Байт має декілька інтерпретацій, можливо використання байтів у якості:

- ◇ цілих чисел у діапазоні 0..255;

- ◇ цілих чисел зі знаком у діапазоні -128..127 (у такому випадку числа представлені у додатковому коді);
- ◇ десяткових цифр у діапазоні 0..9;
- ◇ двійково-десяткових кодів;
- ◇ однобайтових ASCII символів.

У форматі байтових чисел зі знаком старший біт числа указує знак, таке ж правило зберігається і у інших форматах чисел зі знаком.

Слово, що має довжину 16-біт може містити два байти одного з вищенаведених форматів, або використовуватись як єдиний операнд що має лише три інтерпретації:

- ◇ 16-бітне число без знака у діапазоні від 0 до 65535;
- ◇ знакове число від -32768 до +32767 у додатковому коді;
- ◇ 16-бітний Unicode символ (використовується лише командами обробки стрічок).

Подвійне слово може визначатись одним із наведених вище типів даних, але як єдине 32-розрядне число може мати лише дві можливі інтерпретації:

- ◇ 32-бітне знакове число від 0 до 4 294 967 295 ($2^{32}-1$);
- ◇ 32-бітне знакове число від -2^{31} до $2^{31} - 1$.

У МП i386 існують команди обробки стрічок із 32 бітних символів, проте такі символи ще не знайшли застосування.

Числа зі знаком можуть також використовуватись у якості зміщення у командах, що адресують дані та команди у пам'яті.

11.2. Загальний формат команди МП

Загальний формат команди МП зображений на рис. 41. Формат будь-якої команди може містити до 8 полів та мати максимальну довжину до 16 байтів. Перша стрічка загального формату команди зберігає префікси команди. Вони визначають ту чи іншу зміну виконання команди. Друга стрічка є основною та визначає операцію, що виконує команда. Третя стрічка визначає адресу операндів команди та необхідні константи.

Поле Rep/Lock призначене для організації циклічного повторення виконання команди або для забезпечення монопольного володіння шиною МП.

Поле Addr. Size використовується для зміни типу адресації пам'яті та дозволяє використати 32-бітні зміщення для доступу до даних по командам розміщеним у 16-бітних кодових сегментах, або 16 бітні зміщення у 32-бітних кодових сегментах.

Rep/Lock	Addr. Size	Data Size	Seg
0, або 1 байт	0, або 1 байт	0, або 1 байт	0, або 1 байт

Code	Mod Reg R/M	SIB
1, або 2 байти	0, або 1 байт	0, або 1 байт

Rel Addr.	Data
0, 1, 2, або 4 байти	0, 1, 2, або 4 байти

Рис. 41. Формат команди МП i386

Поле Data Size використовується для зміни довжини даних та дозволяє використати 32-бітні регістри та операнди для команд розміщених у 16-бітних кодових сегментах, або 16-бітні регістри у 32-бітних кодових сегментах.

Поле Seg призначене для визначення сегмента, що використовується у команді для адресації даних у пам'яті.

Поле Code визначає код команди. Це поле є основним і може займати 1 чи 2 байти.

Поле Mod Reg R/M визначає тип використовуваних у команді регістрів.

Поле SIB задає масштабний множник для індексного компонента, та назву регістрів, що використовуються у якості індексного та базового. Поле використовується лише при роботі з 32-бітними кодовими сегментами.

Поле Rel. Addr. визначає зміщення, що задане у команді. Зміщення задане у команді може бути як 8 так і 16 чи 32 бітним. Можливий варіант коли зміщення відсутнє. У будь-якому випадку зміщення вважається числом зі знаком заданим у додатковому коді.

Поле Data задає дані, що використовуються у команді. Дані можуть бути 8, 16 чи 32 бітними числами.

Команди використовують усі поля досить рідко. Наприклад, префікси команди відсутні у більшості випадків використання команд. Поле Data використовується лише при операціях із константою, а поле Rel. Addr. лише при деяких операціях із пам'яттю.

11.3. Команди пересилання даних

Ці команди забезпечують передавання даних між регістрами, пам'яттю, портами та стеком.

Команда MOV пересилає дані з одного операнда в інший зі збереженням вмісту першого операнда. Команда XCHG здійснює обмін вмісту двох регістрів або регістра і пам'яті. Команди MOV та XCHG використовується для обміну даними між регістрами МП та між регістрами та комірками пам'яті.

Команда завантаження ефективної адреси LEA призначена для збереження у регістрі МП адреси операнда, що визначається за допомогою зміщення, базової та індексної адресації. Команда призначена для визначення зміщення від початку сегмента елементів складних масивів даних.

Команди завантаження повного вказівника LDS, LES, LFS, LGS та LSS завантажують у регістри МП логічну адресу структури у пам'яті. Сегментна частина адреси записується у регістр DS, ES, FS GS або SS.

Команди розширення розрядності даних призначені для пересилання даних з одночасним розширенням їх розрядності із урахуванням знаків. До таких команд відносять команди CBW, CWD, CWDE, CDQ, MOVSX, MOVSZX. Команди CBW, CWD, CWDE, CDQ, MOVSX розширюють число розміщене у регістрах МП чи у пам'яті, до нового значення з урахуванням знака. Фактично, команди заповнюють старші біти приймача значенням старшого біта джерела даних. Регістри, що використовуються командами та їх розрядності показані у таблиці 17. У таблиці використовуються позначення: r1, r2 – будь-які регістри МП, m – комірка пам'яті.

Команда MOVSZX працює аналогічно команді MOVSX, проте, розширює число заповнюючи старші розряди нулем.

Таблиця 17. Операнди команд розширення розрядності

Команда	Джерело даних	Приймач даних
CBW	AL	AX
CWD	AX	DX:AX
CWDE	AX	EAX
CDQ	EAX	EDX:EAX
MOVSX r1, r2	r2	r1
MOVSX r, m	r1	m

Команди LAFH та SAFH призначені для завантаження та збереження значення молодшого байта регістра ознак. Команда LAFH завантажує у молодший байт регістра ознак значення регістра AH. Команда SAFH зберігає молодший байт регістра ознак у регістрі AH.

Команди IN та OUT призначені для обміну даними між портами вводу-виводу та регістрами МП. Використані у команді дані завжди розміщуються у регістрі AL, AX або EAX. Адреса порта вводу-виводу може бути задана або у регістрі DX або у вигляді константи записаної у коді команди. Значення адреси заданої константою у команді не може перевищувати 0FFh.

Для обміну даними з стеком використовують команди PUSH, POP, POPA, PUSHA, POPFD, PUSHFD. Стек мікропроцесора i386 є стеком слів, або подвійних слів залежно від біта B у дескрипторі сегмента стека. Розрядність регістрів з котрими працюють команди роботи зі стеком визначається дескриптором сегмента кода. У випадку необхідності розрядність даних з котрими працюють команди роботи зі стеком може бути змінена за допомогою префікса перекриття розміру операнда.

Команда PUSH копіює операнд в вершину стека, POP – повертає операнд зі стеку в регістр або комірку пам'яті. Команда PUSHA та PUSHAD – копіює вміст всіх регістрів загального призначення в стек, а POPA та POPAD – повертає вміст усіх 8 регістрів загального призначення зі стека. Команди PUSHA POPA та оперують 16-бітними регістрами, а PUSHAD та POPAD з 32-бітними. Команди POPF, POPDF, PUSHF, PUSHDF виконують дії команд POP, POPD, PUSH, PUSHD над вмістом регістра ознак.

11.4. Арифметичні команди

Арифметичні команди виконують арифметичні дії над двійковими чи десятковими числами.

Команди ADD та ADC виконують операцію додавання. Аргументами команди можуть бути регістри загального призначення, комірки пам'яті та константи. Команда ADD додає два операнди, розміщуючи результат на місці першого з них. Команда ADC носить назву додавання із урахуванням перенесення, вона виконує дії команди ADD та додає до суми значення біта CF регістра ознак.

Команди SUB та SBB виконують операції віднімання. Аргументами команди можуть бути регістри загального призначення, комірки пам'яті та константи. Команда SUB віднімає від першого операнда другий, та розміщає результат на місці першого з них. Команда SBB носить назву віднімання із урахуванням перенесення, вона виконує дії команди SBB та віднімає від суми значення біта CF регістра ознак.

Команди MUL та IMUL виконують операцію множення. Команда MUL виконує операцію множення вважаючи числа додатними. У коді команди задається один операнд, інший операнд береться із регістра AL, AX, чи EAX так, щоб розрядність операндів була однаковою. Якщо операнди мають 8 розрядів, результат розміщується у регістрі AX. Якщо операнди мають 16 розрядів – у регістрах DX та AX, якщо 32 розряди – у регістрах EDX:EAX.

Команда IMUL виконує операцію множення вважаючи числа поданими у додатковому коді. У коді команди може використовувати один, два чи три операнди. Якщо у коді команди заданий один операнд інший множник береться із регістра AL, AX, чи EAX так, щоб розрядність операндів була однаковою. Якщо операнди мають 8 розрядів, результат розміщується у регістрі AX. Якщо операнди мають 16 розрядів – у регістрах DX та AX, якщо 32 розряди – у регістрах EDX:EAX. Якщо у коді команди задані два операнди результат множення першого операнда на другий розміщується на місці першого. Якщо у

кодi команди заданi три операнди результат множення другого операнда на третiй розмiщається у першому операндi. Нажаль, лише константа може бути використана у якостi третього операнда команди IMUL.

Команди DIV та IDIV виконують операцiю дiлення. Команда DIV виконує операцiю дiлення над додатними числами, а команда IDIV над числами зi знаком, поданими у додатковому кодi. У кодi команди задається лише дiльний, iнший операнд береться з регiстрiв AH:AL, DX:AX, чи EDX:EAX. Правила розмiщення результатiв дiлення наведенi у таблицi 18. Остача вiд дiлення завжди має знак, числа, що дiлилось.

Таблиця 18. Розмiщення операндiв команд DIV та IDIV

Розмiр дiльника	Число, що дiлиться	Результат дiлення	Остача
байт	AX	AL	AH
слово	DX:AX	AX	DX
подвiйне слово	EDX:EAX	EAX	EDX

Якщо результат дiлення перевищує максимальне значення, що може бути записане у регiстр, або дiльний рiвний нулю генерується переривання INT 0.

Команди INC та DEC призначенi для збiльшення та зменшення операнда на 1. На вiдмiну вiд команд додавання чи вiднiмання константи, команди INC та DEC не змiнюють ознаку перенесення у регiстрi ознак. Аргументом команди може бути комiрка пам'ятi або регiстр МП.

Команда NEG призначена для змiни знака операнда, який розташований у регiстрi або пам'ятi.

Команда CMP аналогiчна командi вiднiмання SUB, але отриманий результат вiднiмання вiдкидається, встановлюються лише ознаки результата у регiстрi ознак. Команда призначена для порiвняння чисел перед виконанням команд умовного переходу.

МП i386 дозволяє виконувати дiї з числами у ASCII та двiйково-десяткових кодах. Для роботи з числами у кодi ASCII використовують команди корекцiї результата: AAA, AAS, AAM, AAD. Першi три команди мають бути викликанi пiсля арифметичної команди, а остання – до неї.

Команда AAA використовується для ASCII-корекції регістра AX після операції додавання. Якщо молодші 4 біти AL мають значення більше 1001, або біт AF установлений в 1, то $AL = AL + 6$, $AH = AH + 1$, $CF = 1$, $AF = 1$; у протилежному випадку біти CF та AF обнулюються. Старші чотири біта регістра AL у будь-якому випадку обнулюються.

Команда AAS. Виконується після операції ASCII-корекції регістра AX після операції віднімання. Якщо молодші 4 біти AL набули значення більше 1001, або біт AF установлений в 1, то $AL = AL - 6$, $AH = AH - 1$, $CF = 1$, $AF = 1$; у протилежному випадку біти CF та AF обнулюються. Старші чотири біта регістра AL у будь-якому випадку обнулюються.

Команда AAM виконує ASCII-корекцію для арифметичної операції множення. Команда переводить число із двійкової форми у ASCII код. Команда ділить вміст регістра AL на 10, остачу від ділення записує у регістр AL, а результат – у регістр AH. Команда AAD виконує дії протилежні діям команди AAM. У результаті виконання команди регістр AL набуває значення $AL = AL + AH \cdot 10$. Команди AAM та AAD можна використовувати для переводу чисел у двійковий та двійко-десятковий код.

Команди DAA, DAS використовуються для десяткової корекції результату додавання та віднімання. Команди працюють аналогічно командам AAA та AAS, проте замість регістра AH змінюється старші чотири біта регістра AL.

11.5. Команди обробки бітів

Команда XLAT призначена для перекодування байтів за допомогою таблиці. Команда виконує дію:

$$AL = \text{Mem}[\text{DS}:(\text{BX} + \text{AL})], \text{ або } AL = \text{Mem}[\text{DS}:(\text{EBX} + \text{AL})],$$

де $\text{Mem}[\text{Adr}]$ вміст комірки пам'яті з адресою Adr.

До групи логічних команд відносять команди AND, OR, XOR, TEST, NOT. Команди AND OR, XOR призначена для проведення логічних операцій над вмістом регістрів та комірок пам'яті. Результат записується на місце

першого операнда, у якості другого операнда може бути використана константа, або регістр, або комірка пам'яті. Команди виконують дії над 8-, 16- або 32-розрядними операндами. При виконанні команди AND кожний біт результату встановлюється в 1, якщо відповідний біт кожного із операндів рівний 1, у протилежному випадку біт установлюється у нуль. Команда OR встановлює біт результату в 1, якщо відповідний біт хоча б одного із операндів рівний 1, у протилежному випадку – у нуль. При виконанні команди XOR кожний біт результату встановлюється в 1, якщо відповідний біт в обох операндах має протилежне значення, якщо біти приймають однакове значення біт скидається. Залежно від результату операції змінюються значення регістра ознак.

Дії команди TEST аналогічні діям команди AND, але результат виконання команди МП не запам'ятовує. Змінюється лише значення регістра ознак. Команда NOT працює лише з одним операндом. У результаті виконання команди значення кожного біта змінюється на протилежне.

Команди зсуву та циклічного зсуву призначені для переміщення розрядів у середині регістра чи комірки пам'яті. Команди зсувів зсувають перший операнд на число бітів задане константою у другому операнді, або на число задане у регістрі CL. Останній біт, що був висунутий записується у біт CF регістра ознак.

Команди логічних зсувів SHR та SHL призначені для проведення операцій зсуву без організації циклічної передачі даних. Звільнені при зсуві місця заповнюються нулями. Команда SHR зсуває біти вправо, SHL вліво.

Команди арифметичного зсуву SAR та SAL призначені для зсуву двійкових чисел у додатковому коді. Єдина відмінність команди SAR від команди логічного зсуву SHR, що старший біт числа зберігає попереднє значення. Команда арифметичного зсуву вліво повністю аналогічна команді логічному зсуву.

У командах циклічного зсуву ROR, ROL біти даних не втрачаються: біт, що "висувається" зі старшого чи молодшого розряду займає місце, що

звільнилося. Команда ROR здійснює циклічний зсув вправо, ROL – вліво.

У командах циклічного зсуву RCL та RCR біти зсуваються через ознаку CF. Біт, що "висувається", займає місце ознаки CF перенесення, а попереднє значення ознаки CF переноситься на місце, яке звільняється. Команда RCL здійснює зсув вліво, RCR – вправо.

Команди SHRD та SHLD значно відрізняються від попередніх. Вони використовують три операнди: джерело, отримувач та лічильник зсувів. Отримувач зсувається, а біти, які "висуваються", втрачаються і звільнені місця заповнюються бітами джерела. Саме джерело не змінюється. Третім операндом може біти значення регістра CL або задана у команді константа.

Команда встановлення регістра SETxx використовуються для зберігання результатів порівняння. Значення "xx" визначає певне значення бітів регістра ознак. Якщо значення бітів регістра ознак співпадає з заданим в умові xx, всі біти регістра або комірки пам'яті вказані у команді встановлюються в "1", якщо ж співпадіння немає біти встановлюються в "0".

Команди тестування біт BT, BTS, BTR, BTC команди тестування бітів визначає стан біта першого операнда. Номер потрібного біта визначається другим операндом. Команди тестування встановлюють ознаку CF залежно від стану біта. Команда BT не виконує більше ніяких дій. Три інші команди виконують ту ж функцію, але додатково змінюють значення біта. Команда BTS встановлює протестований біт в 1, BTR – скидає, а BTC – інвертує його значення.

Команди сканування BSF, BSR призначені для пошуку ненульових розрядів у даних. Команди приймають два операнди. У другому операнді записується регістр, чи комірка пам'яті, що сканується. У перший операнд буде записаний номер першого біта, що має одиничне значення. Команда BSF шукає першу одиницю в першому операнді, опитуючи біти починаючи від молодшого біта, а команда BSR виконує дію команди BSF починаючи від старшого біта першого операнда.

11.6. Команди обробки стрічок

Команди обробки стрічок зазвичай використовуються у циклах, або з префіксами повторення. В командах обробки стрічок вміст індексного реєстра, що використовується, автоматично змінюється на довжину елемента даних, що обробляються. Якщо біт DF у реєстрі ознак рівний 0, після виконання команди вміст індексних реєстрів, що використовувались командою, збільшується на довжину елемента даних, якщо біт DF рівний 1 – значення у реєстрах зменшуються.

Команда перенесення даних із стрічки в стрічку MOVS (MOVSB, MOVSW, MOVSD) копіює байт, слово чи подвійне слово із комірки пам'яті з адресою DS:SI (DS:ESI) у комірку пам'яті з адресою ES:DI (ES:EDI). Команда змінює вміст реєстрів SI (ESI) та DI (EDI). Команда допускає зміну сегмента для джерела даних.

Команда завантаження у реєстр елемента строки LODS (LODSB, LODSW, LODSD) використовується для зчитування елементів стрічки. Команда LODSB завантажує елемент строки у реєстр AL, LODSW – у AX, LODSD – у EAX. Для адресації строки використовується індексний реєстр SI (ESI). Сегмент задається реєстром DS, можливе перекриття сегмента.

Команда STOS (STOSB, STOSW, STOSD) використовується для збереження значення реєстра у стрічку. Команда STOSB зберігає у стрічці значення реєстра AL, STOSW – AX, STOSD – EAX. Для адресації строки використовується індексний реєстр DI (EDI). Сегмент задається реєстром ES, перекриття сегмента неможливе.

Команда порівняння стрічок CMPS (CMPSB, CMPSW, CMPSD) порівнює байти, слова чи подвійні слова із комірок пам'яті з адресами DS:SI (DS:ESI) та ES:DI (ES:EDI). Команда змінює стан бітів реєстра ознак. Команда допускає зміну сегмента DS на інший.

Команда пошуку елемента SCAS (SCASB, SCASW, SCASD) використовується для пошуку елемента у стрічці. Команда SCASB зчитує байт

з пам'яті за адресою ES:SI (ES:ESI), та віднімає його від регістра AL. Результат віднімання відкидається, відбувається лише встановлення бітів регістра ознак. Команда SCASW працює аналогічно, але зчитує слово і порівнює його з AX, а команда SCASD зчитує подвійне слово та порівнює його з EAX.

Команда читання із порта у стрічку INS (INSB, INSW, INSD) використовується для зчитування даних з порта вводу-виводу та збереження їх у стрічки. Команда INSB зберігає у стрічці значення зчитане з 8-бітного порта, INSW – значення 16-бітного порта, INSD – 32-бітного. Для адресації строки використовується індексний регістр DI (EDI). Адресу порта вводу-виводу задає регістр DX. Сегмент задається регістром ES, перекриття сегмента неможливе.

Команда виведення даних у порт зі стрічки OUTS (OUTSB, OUTSW, OUTSD) використовується для виведення даних зі стрічки у порт. Команда OUTSB виводить у 8-розрядний порт дані, OUTSW – у 16-розрядний, OUTSD – у 32-розрядний. Для адресації строки використовується індексний регістр SI (ESI). Сегмент задається регістром DS, можливе перекриття сегмента. Адреса порта вводу-виводу задається регістром DX.

11.7. Команди передавання керування та організації циклів

До групи відносять команди переходів, виклику та повернення з підпрограм, команди організації циклів та префікси повторення. Команди переходів поділяють на команди умовних та безумовних переходів.

Команди переходів, виклику підпрограм та повернення з підпрограм діляться на дві групи. Команди першого типу здійснюють перехід у середині сегмента, команди другого типу змінюють значення регістра CS та носять назву команд довгих переходів, викликів та повернень. У захищеному режимі роботи передавання керування на сегмент кода можливо лише при однаковому рівні привілеїв, або у випадку вкладки сегмента кода, якщо $CPL \geq DPL$ лише на сегменти кода чи на шлюзи. Передавання керування на сегменти кода не змінює рівень привілеїв, коли необхідно змінити рівень привілеїв використовують шлюзи.

Команда переходу JMP передає керування іншій команді, замінивши вміст лічильника команд IP (EIP). Адреса нової команди може бути в тому ж кодовому сегменті або в іншому. Можливі формати команди JMP наведені у таблиці 19. У таблиці використовуються позначення: IP – зміщення команди, що йде за командою JMP; rel8, rel16, rel32 – 8-, 16-, чи 32-бітне зміщення відносно байта, що йде за командою JMP; r16, r32 – вміст 16- чи 32-розрядного регістра загального призначення; m16, m32 – 16- чи 32-розрядне зміщення адреси комірки пам'яті; ms16 – сегмент адреси комірки пам'яті; M16[Addr], M32[Addr] – вміст комірки пам'яті з адресою Addr; prt16 – сегмент, заданий у команді, i16, i32 – 16- чи 32-розрядне зміщення у команді.

Таблиця 19. Вміст CS та IP (EIP) залежно від типа команд переходів

Команда	Вміст регістра CS	Вміст регістра IP (EIP)	Назва
JMP rel8	Не змінюється	IP+rel8	Короткий відносний
JMP rel16	Не змінюється	IP+rel16	Ближній відносний
JMP rel32	Не змінюється	EIP+rel32	Ближній відносний
JMP r16	Не змінюється	r16	Ближній непрямої ре-гістровий
JMP r32	Не змінюється	r32	Ближній непрямої ре-гістровий
JMP m16	Не змінюється	M16[DS:m16]	Ближній непрямої з ко-міркою пам'яті
JMP m32	Не змінюється	M32[DS:m32]	Ближній непрямої з ко-міркою пам'яті
JMP prt16:i16	prt16	i16	Довгий абсолютний пе-рехід
JMP prt16:i32	prt16	i32	Довгий абсолютний пе-рехід
JMP ms16:m16	M16[ms16:m16+2]	M16[ms16:m16]	Довгий непрямої пере-хід
JMP ms16:m32	M16[ms16:m16+4]	M32[ms16:m32]	Довгий непрямої пере-хід

Операндом внутрішньосегментного переходу є 8-, 16- чи 32-розрядне зміщення, тобто зсув нової команди в поточному кодовому сегменті відносно кінця команди JMP. Операндом міжсегментного переходу є сегмент та зміщення, що дозволяє передавати керування в будь-яке місце будь-якого

сегмента.

Команди умовного переходу Jxx передають керування іншій команді залежно від коду xx умови, яка перевіряється. У системі команд МП є набір команд умовних переходів що перевіряють значення бітів регістра ознак. Також є команди (JCXZ та JECXZ) що перевіряють вміст регістра CX або ECX на рівність нулю. Команди умовних переходів, що перевіряють регістр ознак, завжди використовують 8-, 16- чи 32-розрядне зміщення задане відносно кінця команди умовного переходу. Команди, що перевіряють регістр CX, або ECX використовують лише 8-розрядне зміщення. Зміна сегмента при виконанні команд умовного переходу не здійснюється.

Виклик процедур здійснюється за командою CALL. Формати команд викликів підпрограм наведені у таблиці 20.

Таблиця 20. Вміст регістрів МП від формату команди виклику

Команда	Вміст регістра CS	Вміст регістра IP (EIP)	Заноситься у стек
CALL rel16	Не змінюється	IP+rel16	IP (EIP)
CALL rel32	Не змінюється	EIP+rel32	IP (EIP)
CALL r16	Не змінюється	r16	IP (EIP)
CALL r32	Не змінюється	r32	IP (EIP)
CALL m16	Не змінюється	M16[DS:m16]	IP (EIP)
CALL m32	Не змінюється	M32[DS:m32]	IP (EIP)
CALL prt16:i16	prt16	i16	CS та IP (EIP)
CALL prt16:i32	prt16	i32	CS та IP (EIP)
CALL ms16:m16	M16[ms16:m16+2]	M16[ms16:m16]	CS та IP (EIP)
CALL ms16:m32	M16[ms16:m16+4]	M32[ms16:m32]	CS та IP (EIP)

Команди викликів підпрограм зберігають у стеку значення регістра IP (EIP) та значення регістра CS у випадку його зміни. За командами виклику адреса наступної команди заноситься в стек, після чого завантажується лічильник команд (і регістр CS, якщо перехід робиться в інший сегмент). Виклики можуть мати нескінченну вкладеність і рекурсивність, що обмежуються лише розміром стека.

Команди повернення із підпрограм повертають керування основній програмі. Існують команди близького повернення, довгого повернення та

повернення із переривання. Команди близького повернення RET NEAR призначені для повернення із підпрограми у середині сегмента, вони відновлюють із стека лише збережене значення регістра IP (EIP). Необов'язковий параметр, що може бути заданий, визначає число байтів, що видаляються із стека. За командою довгого повернення RET FAR відбувається відновлення зі стека значень регістрів IP (EIP) та CS. У захищеному режимі роботи, якщо привілеї сегментів коду різні команда повернення відновлює значення привілеїв, що мала програма до виклику підпрограми, перемикає стек, а також обнулює селектори недоступних для недоступних сегментів даних.

Для створення циклів, МП i386 володіє командою LOOP rel8. Команда циклу зменшує значення регістра CX (ECX), якщо регістр CX (ECX) не рівний 0 здійснюється перехід за адресою указаною у команді. Адреса у команді задається 8-розрядним зміщенням. Якщо лічильник циклів рівний 0 здійснюється перехід на наступну команду. Команди LOOPZ та LOOPE здійснюють перехід до наступної команди, якщо регістр CX (ECX) рівний нулю, або ZF = 0; команди LOOPNZ та LOOPNE – якщо регістр CX (ECX) рівний нулю, або ZF = 1.

Команди повторення дозволяють виконати функцію команди LOOP лише з одною командою. Команди повторення забезпечують виконання наступної команди вказану кількість разів. Префікс повторення REP викликає виконання команди стільки разів, скільки визначено вмістом регістра CX (ECX).

Префікси REPE та REPZ викликають повторення команди до встановлення у нуль лічильника, або до встановлення ZF = 0. Префікси REPNE та REPNZ діють аналогічно REPZ, але вихід здійснюється після встановлення ZF в 1.

11.8. Команди керування МП

У системі команд МП існують команди, що скидають та встановлюють окремі біти регістра ознак. На окремі ознаки впливають 7 команд. Команда CLD скидає ознаку напрямку DF, команда STD встановлює її. Команда CLI

скидає ознаку дозволу переривань, STI її встановлює. Команда CLC скидає ознаку перенесення CF, CTC встановлює, а команда CMC інвертує її.

Команда WAIT змушує МП зупинити виконання команд до отримання сигналу BUSY. Використовується під час очікування результату від співпроцесора.

Префікс блокування LOCK дозволяє захопити системну шину у монопольне користування на час виконання наступної команди. Використовується для усунення конфліктів процесорів в мультипроцесорних системах. Команда NOP носить назву "порожньої" команди, або команди "немає операції". Використовується при відлагодженні програм та при формуванні дуже коротких затримок. Команда HLT блокує роботу МП до отримання сигналу "скидання", або сигналу переривання.

11.9. Команди керування захистом

До команд керування захистом належать команди, що керують захистом, чи перевіряють його стан.

Команда завантаження регістра глобальної дескрипторної таблиці LGDT завантажує лінійну базову адресу та граничне значення із шестибайтового операнда у регістр GDTR. Команда збереження регістра глобальної дескрипторної таблиці SGDT зберігає значення регістра GDTR у комірках пам'яті заданих у команді. Команди завантаження регістра дескрипторної таблиці переривань LIDT та збереження його SIDT виконують дії команд LGDT та SGDT над умістом регістра IDTR.

Команда завантаження регістра локальної дескрипторної таблиці LLDT завантажує у регістр LDTR селектор локальної дескрипторної таблиці із регістра чи комірки пам'яті, що вказана у команді. Команда збереження регістра локальної дескрипторної таблиці LLDT виконує дію протилежну команді LLDT.

Команда завантаження слова стану машини LMSW та регістра задачі LTR дозволяють завантажити у регістр MSW та регістр TR із комірки пам'яті чи

регістра загального призначення відповідний код. Команди SMSW та STR виконують зворотні дії.

До групи команд керування захистом також відносять команди завантаження даних у службові регістри МП та команди зчитування службових регістрів. Загальне позначення команд: MOV cr, r32 та MOV r32, cr. У позначенні cr – службовий регістр МП, r32 – 32-розрядний регістр загального призначення.

Команда LAR r16, r/m16 зчитує із дескриптора сегмента селектор, котрого заданий другим аргументом поле Access та заносить його у будь-який регістр загального призначення r16, одночасно ZF встановлюється у 1. Існує модифікація команди, що працює із 32-розрядними регістрами та зчитує поля Access та Properties. Якщо дескриптор не може бути прочитаний, команда встановлює ознаку $ZF = 0$ та не змінює значення регістрів МП. Команда LSL виконує аналогічні дії та має подібні модифікації, але у регістр загального призначення заносить значення поля Limit.

Команда налаштування поля RPL селектора має формат: ARPL r/m16, r16. У команді використовують позначення: r/m16 – 16-розрядний регістр чи комірка пам'яті, r16 – регістр МП. Якщо RPL першого аргументу менший ніж в другого, ознака ZF встановлюється в 1, в RPL першого аргументу записується значення з другого. У противному випадку ZF встановлюється в 0, перший операнд не змінюється.

Команда VERR r/m16 перевіряє можливість читання із сегмента з селектором у r/m16. Якщо читання можливе встановлюється ознака $ZF = 1$, у противному випадку ZF скидається. Команда VERW r/m16 перевіряє можливість записування даних у сегмент.

Команда перевірки меж масиву BOUND може бути використана, щоб перевірити, чи індекс масиву знаходиться у межах. Команди ENTER (входу в процедуру) і LEAVE (виходу з процедури) призначені для мов високого рівня. За командою ENTER створюється кадр і образ стека, які використовуються компіляторами для зв'язування викликів процедур. За командою LEAVE кадр

стека й образ видаляються зі стека для підготовки повернення до процедури, яка зробила виклик.

Контрольні питання

1. Мікропроцесори i386: характеристики МП та режими роботи.
2. Захищений режим роботи МП i386.
3. Призначення основних функціональних вузлів МП i386: регістри загального призначення та індексні регістри МП.
4. Призначення основних функціональних вузлів МП i386: сегментні регістри, їх використання в захищеному та реальному режимах роботи.
5. Призначення основних функціональних вузлів МП i386: регістри керування пам'яттю, їх структура та використання.
6. Формування лінійної та фізичної адреси команд та даних в захищеному режимі роботи МП i386, селектори та дескрипторні таблиці, формати дескрипторів.
7. Організація мультизадачності мікропроцесора i386, регістр задач та сегмент стану мікропроцесора.
8. Режим віртуального 86.

Мікропроцесор i486

1. Особливості організації та основні характеристики

Мікропроцесор i486 швидкодіючий 32-бітний МП, призначений для роботи у мультизадачних операційних системах. Цей мікропроцесор повністю сумісний із попередніми моделями i8086, i80286, i386, але в той же час відрізняється більшою швидкодією. Більшість команд даного МП виконуються за один машинний цикл. Наявність на кристалі 8-кілобайтного кеша і забезпечує високу продуктивність МП навіть при використанні відносно повільної пам'яті.

Крім того, МП i486 може містити вбудований арифметичний співпроцесор, який значно прискорює виконання арифметичних команд.

МП i486 майже повністю за будовою повторює МП i386. Відмінності стосуються в основному системного рівня, а точніше, системних регістрів, з якими здебільшого працюють програми операційної системи або драйвери розширеної пам'яті.

МП i486 використовує зарезервовані в МП i386 біти системних регістрів CR0, CR3, а також регістра ознак EFLAGS, є деякі відмінності в дескрипторних таблицях сторінок, а також кілька нових команд.

У регістрі CR0 стали визначеними п'ять нових бітів - NE, WP, AM, NW, CD. У регістрі CR3 визначені нові біти - PCD та PWT.

У регістрі EFLAGS з'явилась нова ознака AC (біт 18). Цей біт разом з бітом AM регістра CR0 контролює вирівнювання об'єкта в пам'яті.

Додалися три команди, які призначені для використання прикладними програмами – BSWAP, XADD, CMPXCHG, а також три нові системні команди, які керують кешем та TLB: INVD, WBINVD, INVLPG. Нові команди використовують мультизадачні операційні системи.

Також було змінено набір сигналів інтерфейсу МП, на відміну від попередників цей процесор перестав потребувати "косого" буфера для роботи з 8-розрядною шиною вводу-виводу. Введена можливість множення тактової

Промисловість випустила кілька модифікацій МП з такими характеристиками:

- ## 2. Формат регістра CR0 i486



106

Таблиця 21. Призначення бітів регістра CR0 процесора i486

Номер та позначення біта		Призначення
0	PE	Ввімкнення захищеного режиму роботи процесора
1	MP	Присутність співпроцесора
2	EM	Емуляція співпроцесора
3	TS	Переключення задачі
4	ET	Тип співпроцесора. Для процесора i486 біт ET = 0 для процесора i486SX, для інших процесорів біт встановлений в 1
5	NE	Числова помилка. Дозволяє опрацьовувати помилки при операціях із плаваючою крапкою
6-15	–	Зарезервовано.
16	WP	Захист запису. При встановленні цього біта сторінки користувача захищені від запису у режимі супервізора
17	–	Зарезервовано
18	AM	Біт маски вирівнювання. Цей біт дозволяє або забороняє контроль вирівнювання операндів команд у пам'яті. Контроль вирівнювання дозволений тільки для програм, що працюють в третьому кільці, за умови встановлення біта AM
19-28	–	Зарезервовано.
29	NW	Дозвіл наскрізного записування даних. Використовується у механізмі керування кешуванням
30	CD	Заборона кешування. Якщо цей біт встановлено, робота внутрішнього кеша заборонена
31	PG	Включення механізму трансляції сторінок

3. Формат регістра CR3 процесора i486

Регістр CR3 змінив формат порівняно із i386 (рис. 43). Старші біти регістра й далі визначають базову фізичну адресу каталога таблиць сторінок.

Біт PWT визначає прозорість запису на рівні сторінок. Він використовується для керування записом у зовнішній кеш.

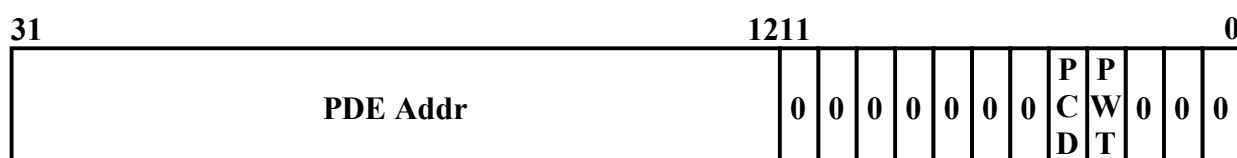


Рис. 43. Формат CR3

Біт PCD забороняє кешування на рівні сторінок. Використовується для керування роботою зовнішнього кеша.

4. Системні команди мікропроцесора i486

Системні команди призначені для використання в основному, в модулях операційних систем (у модулях ядра операційної системи, драйверах і т. п.). Деякі з перерахованих нижче команд застосовують і при розробленні прикладних програм, які працюють у захищеному режимі.

Як правило, системні команди можуть використовувати тільки ті програми, які виконуються у нульовому привілейованому кільці.

Команда ARPL здійснює корекцію поля привілеїв ініціатора запиту в селекторі. Цю команду використовують системні модулі, щоб перевірити рівень запитуваних привілеїв у переданих їм як параметри селекторів. Прикладна програма не має права запитувати привілеї, що перевищують її власні. Перший операнд команди – 16-розрядний регістр або слово в пам'яті, які містять значення селектора, що перевіряється. Другий операнд - регістр, в якому записано вміст CS прикладної програми. Якщо команда не змінювала рівень привілеїв, у регістрі FLAGS (EFLAGS для мікропроцесорів i386 та i486) встановлюється ознака нуля. В іншому випадку ця ознака скидається.

Приклад використання команди:

```
mov dx, cs
mov ax, TESTED_SELECTOR
arpl dx, ax
```

Команда CLTS здійснює скидання ознаки TS переключення задачі в регістрі CR0. Кожного разу при перемиканні задачі ознака TS встановлюється в одиницю. Команда CLTS дозволяє скинути цю ознаку.

Команда LAR завантажує байт прав доступу. Для мікропроцесора i80286 команда LAR завантажує у перший операнд (регістр) байт доступу дескриптора, обраного другим операндом. Другий операнд є селектором, який вказує на використовуваний дескриптор. У мікропроцесорах i386 і i486 команда LAR використовує як перший операнд 32-розрядний регістр. Крім байта прав доступу в цей регістр заносять біти типу сегмента (9-11), DPL (14), біт

присутності (15), біт гранулярності (23).

Команда LGDT завантажує регістр глобальної дескрипторної таблиці GDTR. Команда виконує ініціалізацію регістра GDTR, який вказує на розміщення у пам'яті та розмір глобальної таблиці дескрипторів.

Команда LIDT завантажує регістр дескрипторної таблиці переривань IDTR. Команда виконує ініціалізацію регістра IDTR, який вказує на розташування в пам'яті і розмір дескрипторної таблиці переривань.

Команда LLDT завантажує регістр локальної дескрипторної таблиці LDTR. Команда виконує ініціалізацію регістра LDTR, який вказує на розміщення у пам'яті й розмір локальної таблиці дескрипторів.

Команда LMSW завантаження слова стану мікропроцесора. За допомогою цієї команди виконати завантаження молодшого слова регістра CR0 з регістра або операнда команди. Ця команда може використовуватися для перемикання мікропроцесора в захищений режим. Зворотного перемикання дана команда не забезпечує, навіть для мікропроцесорів i386 і i486.

Команда LSL завантажує межі сегмента. Команда має два операнди. Границя сегмента, селектор якого використовується як другий операнд і заданий у регістрі, завантажується у регістр, зазначений як перший операнд.

Команда LTR завантаження регістра задачі. Команда призначена для завантаження регістра TR - регістра задачі. Завантаження цього регістра не призводить до перемикання задачі.

Команди завантаження системних регістрів MOV. Для мікропроцесорів i386 і i486 як операнди звичайної команди MOV припустимо на нульовому рівні привілеїв вказувати системні регістри: CR0, CR2, CR3, DR0, DR1, DR2, DR3, DR6, DR7, TR6, TR7. Команду MOV можуть використовувати мікропроцесори i386 і i486 для повернення мікропроцесора із захищеного режиму в реальний.

Команда SGDT записує у пам'ять вміст регістра GDTR. Команда дозволяє довідатися про поточний вміст регістра глобальної дескрипторної таблиці GDTR, використовують у системних відлагоджувальних програмах.

Команда SIDT записує в пам'ять уміст регістра IDTR. Команда дозволяє довідатися про поточний вміст регістра глобальної дескрипторної таблиці переривань IDTR, використовується у системних відлагоджувальних програмах.

Команда SLDT записує в пам'ять уміст регістра LDTR. Команда дозволяє довідатися про поточний вміст регістра локальної дескрипторної таблиці LDTR.

Команда SMSW записує слово стану мікропроцесора. Команда записує у пам'ять чи 16-бітовий регістр молодше слово регістра CR0.

Команда STR записує поточний вміст регістра задачі TR у 16-розрядну комірку пам'яті чи 16-розрядний регістр.

Команда VERR перевіряє сегмент на можливість читання, а команда VERW перевіряє сегмент на можливість запису. За допомогою цих двох команд можна перевірити доступ до обраного селектором сегмента на читання і запис. Якщо операція читання або запису доступна, ознака нуля ZF встановлюється в одиницю, в іншому випадку вона скидається у нуль. Основне призначення цієї команди – запобігти появі виключень захисту пам'яті при спробі звертання до сегмента.

Контрольні питання

1. Мікропроцесор i486: характеристики МП та режими роботи.
2. Особливості використання системних регістрів МП i486.
3. Додаткові команди МП i486 в порівнянні з МП i386.
4. Призначення основних функціональних вузлів МП i486: сегментні регістри, їх використання в захищеному та реальному режимах роботи.
5. Призначення основних функціональних вузлів МП i486: регістри керування пам'яттю, їх структура та використання.
6. Формування лінійної та фізичної адреси команд та даних в захищеному режимі роботи МП i486, селектори та дескрипторні таблиці, формати дескрипторів.

Архітектура процесорів Pentium

1. Особливості організації МП Pentium

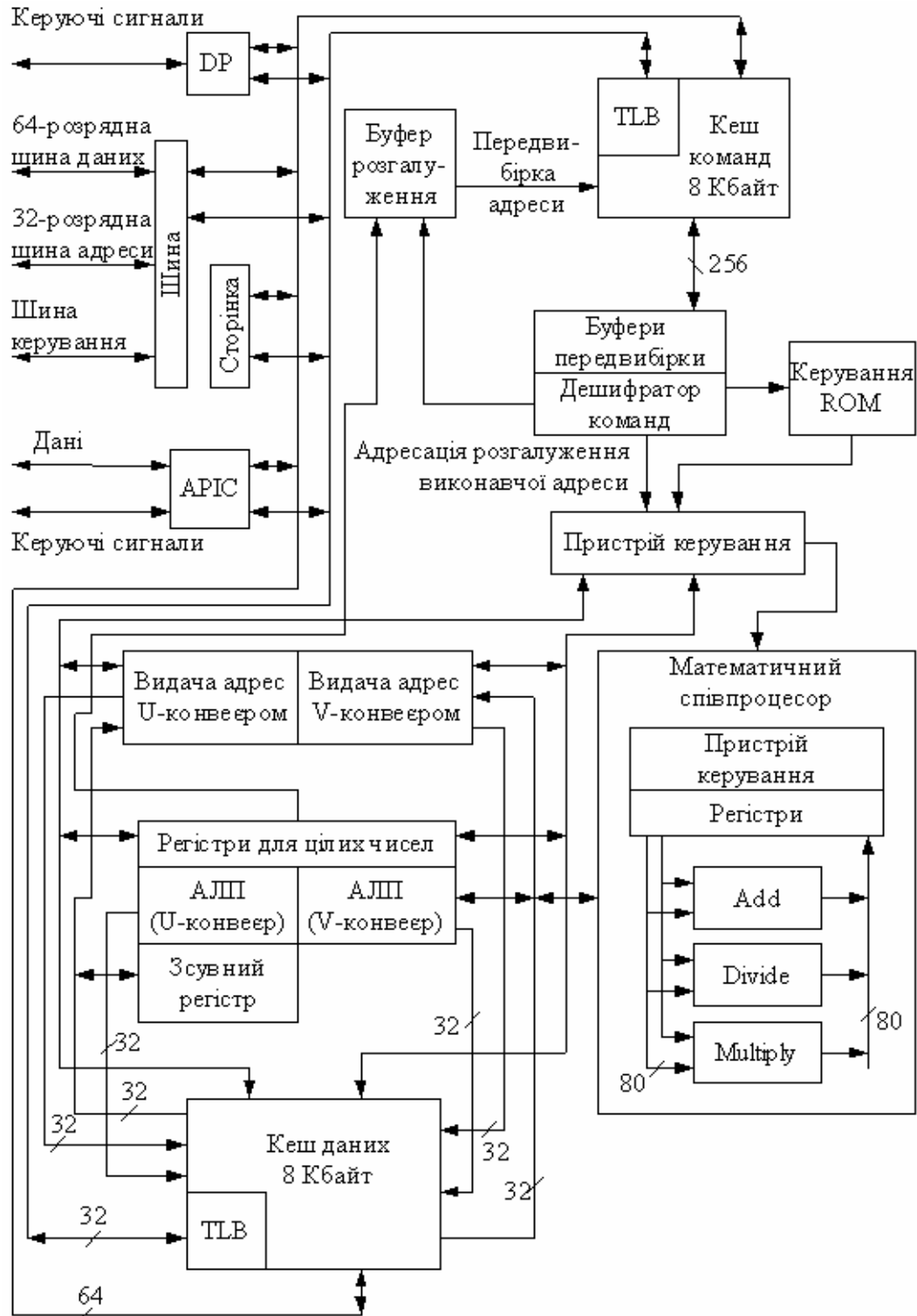


Рис. 44. Структурна схема процесора Pentium

Процесор Pentium (рис. 44) має суперскалярну архітектуру, що дає можливість одночасно виконувати за один такт дві інструкції. Процесор

побудований на основі двох конвеєрів загального призначення і конвеєра співпроцесора для виконання операцій над числами із плаваючою крапкою. Отже процесор може одночасно виконувати дві цілочислові інструкції. Механізм динамічного передбачення переходів мінімізує простої конвеєра під час розгалужень програми.

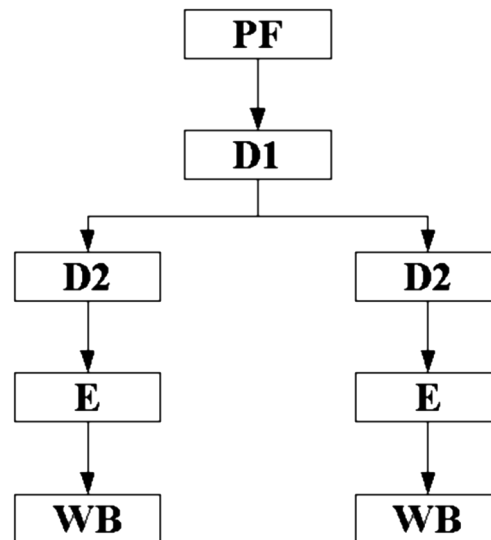


Рис. 45. Структура цілочислового конвеєра Pentium.

Структура цілочислових конвеєрів зображена на рис. 45. Обидва конвеєри мають по п'ять ступенів: PF (Pre Fetch) – попередня вибірка; D1 (Decode stage 1) – перша стадія декодування; D2 (Decode stage 2) – друга стадія декодування; E (Execute) – виконання; WB (Write buffer) – буфер запису.

Функціонально обидва конвеєри подібні, але другий конвеєр (V) має певні обмеження порівняно з першим (U). При виконанні програми одночасно аналізуються дві команди, і якщо одну з них можна виконати на конвеєрі U, то для виконання другої запускаються обидва конвеєри. Якщо жодна з них не може бути виконана на конвеєрі V, то обидві вони послідовно виконуються на конвеєрі U, конвеєр V у цей час простоює. Більшість програм, які часто використовуються, можна впорядкувати таким чином, щоб конвеєри відпрацьовують кожну пару інструкцій за один такт.

Ступінь D2, на якому обчислюються адреси операндів пам'яті, має багатоканальний суматор. Цей ступінь не потребує додаткових тактів затримки при багатокomпонентних обчисленнях адрес – на відміну від конвеєра МП i486.

Вбудована підсистема кешування містить два двоканальні набірно-асоціативні кеші розміром по 8 кбайтів. Довжина рядка кешу – 32 байти, ширина зовнішньої шини даних – 8 байтів. Обидва кеші можуть реалізувати архітектуру зворотного запису, хоча в кеш інструкцій запис не здійснюється, та алгоритм заміщення даних LRU (Least Recently Used). Алгоритм LRU заміщує дані, що були використані найдавніше. Пам'ять процесорного кеша поділена на вісім банків з чергуванням з межами в 4 байти. Кеш даних доступний обом конвеєрам. Потреби у даних із різних банків з обох конвеєрів можуть обслуговуватись одночасно.

Блок попередньої вибірки інструкцій має чотири 32-байтні буфери. На ступені РР дві незалежні пари буферів працюють разом із буфером адрес переходів ВТВ (Branch Target Buffer). Попереднє зчитування інструкцій у кожний окремий момент часу може запитувати лише один буфер. Вибірка виконується послідовно – до появи інструкції розгалуження. При появі розгалуження блок ВТВ передбачає дії команди розгалуження за попереднім виконанням цієї команди. Якщо перехід не передбачається, то триває лінійне зчитування інструкцій, якщо ж передбачається, то дозволяється робота іншого буфера попереднього зчитування, який починає її з точки переходу. Якщо передбачений перехід не здійснився, конвеєри інструкцій очищаються, і передвибірка починається знову. Блок може вибрати інструкції, що займають не більше двох рядків кеша. Конфлікти між блоком попередньої вибірки інструкцій та запитами даних з кеша мінімізовані завдяки розділенню кеша команд та кеша даних.

У процесорі Pentium уведено схему динамічного передбачення переходів із буфером переходів (ВТВ) на 256 входжень. Інструкції виконуються без скидів конвеєра та штрафних циклів, коли передбачення правильне. Якщо перехід передбачено неправильно, то умовний перехід, який здійснюється на конвеєрі V, потребує три штрафні цикли, а на конвеєрі U – чотири. Помилково передбачений виклик підпрограми чи безумовний перехід потребує три штрафні цикли на будь-якому конвеєрі.

Кожний конвеєр має свій буфер запису WB для підвищення продуктивності під час послідовних операцій записування в пам'ять. Буфери 64-розрядні й заповнюються за один такт. На зовнішню шину з цих буферів операції записування надходять у тому ж порядку, в якому вони генеруються ядром процесора, підтримується чіткий порядок запису.

Математичний співпроцесор реалізований на триступеновому конвеєрі з плаваючою крапкою, який побудовано до цілочислового конвеєра, і до ступеня E інструкції FPU проходять на спільному конвеєрі. Після цього за один такт вони проходять ступенями X1, X2 і WF конвеєра FPU. Під час виконання «довгих» інструкцій FPU можуть виконуватись інструкції на цілочисловому конвеєрі. Стек регістрів FPU можна використовувати як звичайний набір регістрів, виконуючи інструкції FXCH, паралельно з виконанням «довгих» обчислювальних інструкцій FPU.

2. Системна шина процесорів Pentium

Системна шина процесорів Pentium – подібна на системну шину процесора i486. Вона удосконалена для досягнення максимальної продуктивності, підтримування кеша зворотного запису та розширення функціональних можливостей. Шина адреси дає змогу адресувати 4 Гбайти фізичної пам'яті. Циклічне згортання адреси для емуляції реального режиму процесора i8086 здійснюється примусовим встановленням нуля на лінії A20. Для виконання інструкцій вводу-виводу передбачено 64 кбайти адресного простору портів (00000000-0000FFFFh).

Лінії A31-A3 ідентифікують адресу з точністю до 64-розрядного слова. У межах цього слова сигнали BE0#-BE7# безпосередньо вказують, які байти використовують у даному циклі. Лінії A31-A5, що визначають адресу рядка внутрішнього кеша, в циклах стеження працюють на введення. Адреса контролюється на паритет. При виявленні помилки паритету генерується сигнал помилки ARCHK#, який опрацьовує системна логіка.

Шина даних процесорів Pentium 64-розрядна, динамічного керування

розрядністю шини немає. Узгодження за розрядністю з інтерфейсними шинами виконують мікросхеми чіпсета. Розряди паритету DP7-DP0 належать до кожного байта шини даних. Схеми паритету генерують контрольні біти в циклах запису, а в циклах зчитування у разі помилки паритету на виході PCNK# генерується сигнал помилки. При контролі паритету, який дозволяється сигналом PEN, МП фіксує «збійну» адресу і дані в регістрі машинного контролю. Якщо контролювання помилок паритету дозволено (в регістрі CR4 встановлений біт MCE) та якщо є помилки генерується особлива ситуація #MC. Пакетні цикли виконуються тільки при зверненнях до кешованої пам'яті. За такого кешування передбачається підтримка пакетного режиму. Конвеєрна адресація, як і в процесорах i80286 і i386, дає змогу одночасної наявності на системній шині двох запитів, що обслуговуватимуться. Без конвеєризації наступна адреса та тип циклу виставляються тільки після завершення передачі даних поточного циклу.

Тип шинного циклу процесор Pentium встановлює типовими для фірми Intel сигналами M/IO#, D/C# і W/R#, що діють одночасно зі стробом ADS#. Готовність зовнішніх схем до завершення циклу підтверджується сигналом BRDY#. Крім циклів звернення до пам'яті, вводу/виводу і підтвердження переривання, процесор Pentium має спеціальні шинні цикли, що ідентифікуються за комбінацією сигналів BE0-BE7 у випадку встановлення M/IO# у стан лог. 0, а сигналів D/C# та W/R – у стан одиниці. Типи спеціальних циклів наведені у таблиці 22.

Таблиця 22. Спеціальні цикли процесора Pentium

BE7-BE0	Тип цикла
11111110	Shutdown – аварійна зупинка
11111101	Flush – очищення кешу (команди INVD, WBINVD)
11111011	Halt – зупинник (команда HALT)
11110111	Writeback – зворотній запис (команда WBINVD)
11101111	Flush Acknowledge – підтвердження очищення (сигнал FLUSH#)
11011111	Branch Trace Message – повідомлення трасування переходів

Усі цикли має підтвердити сигнал BRDY#. Вхід EWBЕ# призначений для

відстеження стану зовнішніх буферів відкладеного запису, чим забезпечується коректна послідовність шинних циклів запису. Для узгодження даних кеша та основної пам'яті процесор Pentium відпрацьовує цикли стеження (Snoop Cycle чи Inquire Cycle), які ініціює зовнішня для процесора система. Такі цикли використовують сигнали ANOLD#, EADS# і відповідні сигнали процесора HIT# і HITM#. Вивантаження усіх модифікованих рядків первинного кеша зумовлює сигнал FLUSH. Цикли стеження ініціює система для визначення наявності ділянки пам'яті, що вимагалась у рядку будь-якої кеш-пам'яті та оцінювання її стану. Процесори Pentium підтримують протокол MESI.

Процесор синхронізується високостабільним генератором, котрий задає частоту шини. Припустимий діапазон частоти шини 33-66 МГц, типові значення частот: 50, 55, 60 і 66 МГц. У середині процесора відбувається множення вхідної частоти. Для процесорів із частотою $75 \div 133$ МГц коефіцієнт множення дорівнює 1,5 або 2. Він визначається сигналом BP. Інші моделі сприймає два сигнали: BP0 і BP1. Значення коефіцієнтів множення частоти K для різних моделей Pentium наведені у таблиці 23. У таблиці частота ядра Fcore вказана для частоти зовнішньої шини 66,6 МГц. Коефіцієнт множення 4 є лише у версії процесора Pentium призначеної для мобільних застосувань.

Таблиця 23. Коефіцієнти множення частоти для різних моделей Pentium

Сигнали	75-133 МГц		166-233 МГц	
	K	Fcore	Kf	Fcore
BF1-BF0				
00	2,5	166	2,5	166
01	3,0	200	3,0	200
10	2,0	133	2,0(4)	133 (266)
11	1,5	100	3,5	233

Процесори сімейства Pentium після закінчення дії сигналу RESET, крім звичайного робочого режиму, можна перевести у режими:

- ◇ BIST (Built In Self Test) – режим самотестування, що охоплює внутрішні блоки процесора (відпрацьовується за 219 тактів ядра). Після цього процесор переходить у робочий режим. Результат відпрацювання тесту заноситься у регістр EAX. Якщо значення цього

регістра нульове то несправностей у процесорі немає, ненульове значення вказує на несправності у певному блоці МП. За наявності внутрішньої помилки паритету генерується сигнал IERR# і процесор намагається задіяти аварійну зупинку (Shutdown). Тест BIST запускається під час спадання сигналу RESET при високому рівні сигналу INIT.

◇ Tristate Test Mode – режим переведення усіх вихідних та двонаправлених ліній у високоімпедансний стан. Перехід у цей режим здійснюється за низького рівня сигналу FLUSH# під час спадання сигналу RESET.

◇ FRC – режим, у якому процесор працює як перевіряючий у двопроцесорній системі з надлишковим контролем функціонування. Перехід до цього режиму здійснюється за низького рівня на вході FRCMC# під час спадання сигналу RESET.

Апаратні переривання процесорів Pentium здійснюються сигналами (пріоритет сигналів наведено у порядку спадання):

- ◇ BUSCHK# – контроль шини, що зумовлює особливу ситуацію #MC;
- ◇ R/S# – переведення у зондовий режим;
- ◇ FLUSH# – очищення кеш-пам'яті;
- ◇ SMI# – переривання входження у режим SMM;
- ◇ INIT – "м'яке" скидання процесора;
- ◇ NMI – немасковане переривання;
- ◇ INTR – запит маскованих переривань;
- ◇ STOPCLK# – переведення в енергозберігаючий режим з виключенням тактової частоти.

Переривання у даному випадку стосуються усіх подій, які змушують процесор генерувати зовнішні цикли позачергово. Якщо дозволена робота контролера APIC, то переривання, які надходять шиною процесора, замінюють сигнали NMI і INTR. У процесорах Pentium 2-го покоління можна змінювати

порядок пріоритету переривань за допомогою розряду ITR регістра TR12. У таблиці 24 наведені два можливі варіанти пріоритетів.

Таблиця 24. Можливі варіанти пріоритету переривань

Пріоритет переривання	ITR=0 (за замовчуванням)	ITR=1
1	Точка зупинки (INT3)	Точка зупинки (INT3)
2	BUSCHK#	BUSCHK#
3	Пастка відлагодження INT1	FLUSH#
4	R/S#	SMI#
5	FLUSH#	Пастка відлагодження INT1
6	SMI#	R/S#
7	INIT	INIT
8	NMI	NMI
9	INTR	INTR
10	Помилка FPU	Помилка FPU
11	STOPCLK#	STOPCLK#
12	Відмова на наступній команді	Відмова на наступній команді

Крім того, існує можливість рестарту команд у режимі SMM, а також зниження енергоспоживання у неробочому режимі. За сигналом STOPCLK# процесор розвантажує буфери записування даних і входить у режим Stop Grant, в якому скорочується тактування більшості вузлів процесора. Це зумовлює зниження енергоспоживання приблизно в 10 разів. У цьому стані процесор скорочує виконання команд і не обслуговує переривань, але продовжує слідкувати за шиною, відстежуючи кеш-потрапляння. З цього стану процесор виходить після зняття сигналу STOPCLK#. Використання режиму SMM разом із керуванням сигналом STOPCLK# реалізує механізм розширеного керування живленням APM (Advanced Power Management).

У режим зниженого енергоспоживання Auto HALT Power Down процесор переходить, виконуючи команду HALT. У цьому стані він реагує на всі переривання і продовжує стежити за шиною. У режимі зупинення зовнішньої синхронізації процесор споживає мінімальну потужність (менше 0,3 Вт). Наступне подавання синхронізації супроводжує сигнал RESET.

Розширення архітектури процесорів Pentium порівняно з базовою архітектурою 32-розрядних процесорів покоління зводиться до додавання

додаткового наборів регістрів, специфічних для моделі MSR (Model Specific Registers), зокрема група тестових регістрів (TR1...TR12), засобів моніторингу продуктивності, регістрів-фіксаторів адреси і даних циклу, який спричинив спрацювання контролю машинної помилки. Ці регістри можуть бути несумісними для різних класів (P5 і P6) і навіть для різних моделей процесорів одного класу. У програмі, що базується на застосуванні даних регістрів, використовують відомості про процесор, отримані за допомогою команди CPUID.

Контрольні питання

1. Мікропроцесор Pentium: характеристики МП та режими роботи.
2. Паралельна обробка даних у МП Pentium.
3. Додаткові команди МП.
4. Тактові частоти та множення частоти у МП Pentium.
5. Призначення основних функціональних вузлів МП.
6. Формування лінійної та фізичної адреси команд та даних в захищеному режимі роботи МП, селектори та дескрипторні таблиці, формати дескрипторів.

Мікро-ЕОМ сімейства МК48

1. Сімейство однокристальних мікро-ЕОМ

Однокристальні мікро-ЕОМ призначені для виконання нескладних операцій керування та регулювання різноманітних об'єктів. На відміну від універсальних процесорів, однокристальні мікро-ЕОМ (ОМЕОМ) містять своєму складі усі, чи більшість функціональних блоків "звичайної" ЕОМ.

Зазвичай, однокристальні мікро-ЕОМ (ОМЕОМ) випускають у кількох модифікаціях, котрі відрізняються типом та об'ємом запам'ятовуючого пристрою, числом портів вводу-виводу, максимальною тактовою частотою та енергоспоживанням.

Більшість типів ОМЕОМ мають модифікацію з вбудованим репрограмованим ПЗП з ультрафіолетовим чи електричним стиранням, або можливістю підключення зовнішнього ПЗП програм. Такі прилади, крім процесу відлагодження, можна використовувати як самостійні вироби в умовах, коли замовлення ОМЕОМ з масочним ПЗП недоцільне з якихось причин.

Залежно від структури пам'яті виділяють два типи архітектур ОМЕОМ. Якщо використовується єдина пам'ять для розміщення програм і даних, і при цьому будь-яка комірка може бути використана як для розміщення коду операції, так і для розміщення будь-яких даних, у тому числі результату операції, то таку архітектуру МП називають нейманівською (прістонською). Її використовують деякі сімейства ОМЕОМ, наприклад, нейманівську архітектуру використовує сімейство ST7 фірми STMicroelectronic. Проте, найчастіше вона використовується для побудови ЕОМ загального призначення, оскільки дозволяє при даному обсязі пам'яті здійснити оптимальне розміщення даних та програм у пам'яті. Основним недоліком нейманівської архітектури є використання однієї і тієї ж шини для доступу як до кодів команд, так і до даних, що за інших рівних умов збільшує час доступу до пам'яті.

Якщо ж в доступ до даних та програм організований по різних шинах, а пам'ять даних та пам'ять програм розділені фізично і логічно, то такі мікро-

ЕОМ відносять до гарвардської архітектури. Гарвардська архітектура забезпечує вищу продуктивність звертань до пам'яті, однак при цьому накладає додаткові обмеження на обсяг програми та даних. Основний напрямок використання такої архітектури – спеціалізовані ЕОМ, що вбудовуються в апаратуру. Більшість сімейств однокристальних мікро-ЕОМ будується на основі цієї гарвардської архітектури.

Сімейство однокристальні мікро-ЕОМ МК48 складається із мікросхем кількох поколінь, виконаних по різних технологіям. Основні характеристики типових представників сімейства наведені у таблиці 25.

Таблиця 25. Основні характеристики мікросхем сімейства МК48

Мікросхема	Аналог	Внутрішня пам'ять програм	Тип ПЗП програм	Внутрішня пам'ять даних байт	Максимальна робоча частота МГц	Споживаний струм мА	Технологія виготовлення
KP1816BE35	i8035	–	зовнішня	64	6	135	n-МОП
KP1816BE48	i8748	1 кбайт	УФ EPROM	64	6	135	n-МОП
KP1816BE39	i8039	–	зовнішня	128	11	110	n-МОП
KP1816BE49	i8049	2 кбайти	PROM	128	11	110	n-МОП
KP1830BE35	i80C35	–	зовнішня	64	6	8	КМОП
KP1830BE48	i80C48	1 кбайт	PROM	64	6	8	КМОП

2. Призначення виводів мікросхеми

Розгляд сімейства проведемо на прикладі мікросхеми KP1816BE48. Умовне позначення однокристальної мікро-ЕОМ KM1816BE48 зображено на рис. 46, призначення виводів наведено у таблиці 26. Лінії портів P1 та P2 є квазідвонаправленими тобто працюють як виводи з відкритим колектором, котрі мають внутрішні резистори, що підтягують лінію до високого рівня. Тому при встановленні відповідного біта порту в стан лог. 1 вивід видає невисокий струм, а при встановленні в стан лог. 0 може приймати досить значний струм. Така асиметрія полегшує узгодження виводу з входами і виходами ТТЛ мікросхем. Зауважимо, що скидання мікросхеми за сигналом $\overline{\text{CLR}}$ записує у всі розряди портів P1 та P2 одиницю, а також перемикає лінію T0 на введення даних.

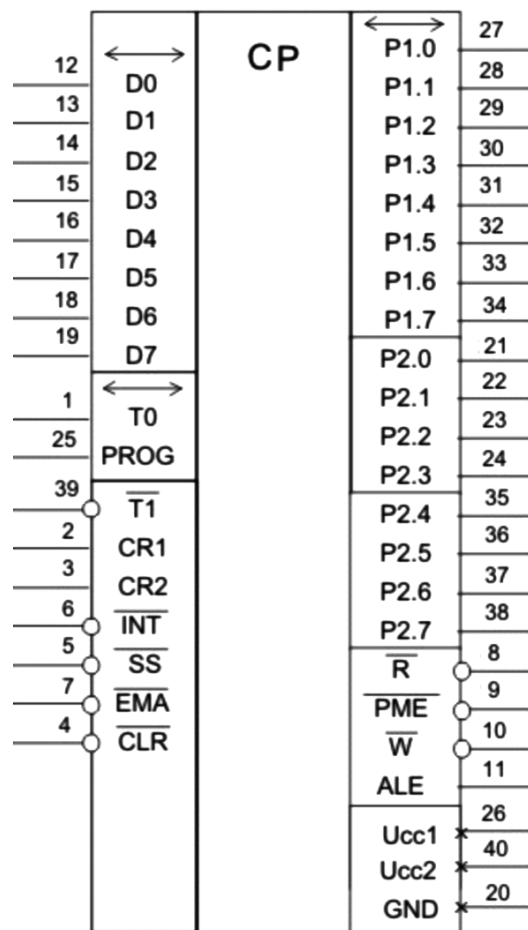


Рис. 46. Умовне позначення KM1816BE48

Таблиця 26. Призначення виводів мікросхеми

Позначення	Номер	Призначення
1	2	3
GND	20	Загальний
U _{CC1}	26	Напруга живлення ОЗП у режимі нормальної роботи. У режимі програмування – напруга програмування
U _{CC2}	40	Напруга живлення решти блоків мікросхеми. Завжди дорівнює 5В
$\overline{\text{CLR}}$	4	Сигнал скидання мікросхеми. Логічний нуль призводить до встановлення мікросхеми в початковий стан
$\overline{\text{INT}}$	6	Вхід маскованого запиту переривання від зовнішнього джерела, може опитуватись командою JNІ
PROG	25	У режимі програмування – це вхід для подавання програмуючого імпульсу (+25 В). В нормальному режимі роботи використовується для подавання сигналу синхронізації для розширювача вводу-виводу
PME	9	Строб вибірки з пам'яті програм. Логічний нуль вказує, що відбувається вибірка коду команди з зовнішньої пам'яті

Продовження таблиці 26

1	2	3
CR1, CR2	2, 3	Входи під'єднання кварцового резонатора, або резонатора іншого типу. Також на вхід X1 KM1816BE48 можна подавати сигнал синхронізації від зовнішнього джерела
$\overline{SS}$	5	Покрокове виконання. Логічний нуль на лінії призупиняє виконання кожної команди на етапі видавання адреси для вибірки коду команди до тих пір, поки на лінію SS не надійде логічна 1. При цьому на лінії порту P2 та P0 виводиться адреса комірки пам'яті програм, код з якої має бути вибраний. Сигнал дозволяє виконувати програму з зупинкою після виконання кожної чергової команди
ALE	11	Стробуючий сигнал адреси зовнішньої пам'яті. Використовується для вказання наявності та фіксації адреси зовнішньої пам'яті на зовнішньому регістрі. Логічна одиниця вказує на наявність на лініях порту P0 та P2 нової адреси. Фіксація адреси повинна відбуватись при переході сигналу з високого рівня в низький. Сигнал формується в кожному машинному циклі і може використовуватись в якості сигналу синхронізації з частотою, що дорівнює 1/15 частоти тактового сигналу
EMA	7	Сигнал виключення внутрішнього ПЗП програм. Лог. 1 на вході вказує МК на необхідність виконувати вибірку всіх команд із зовнішньої пам'яті. Використовується з метою налагодження та роботи з модифікаціями контролера, що не мають внутрішньої пам'яті програм
R	8	Читання. Логічний нуль вказує, що відбувається читання зовнішньої пам'яті даних, або читання порта D0 – D7
W	10	Записування. Логічний нуль вказує, що відбувається записування зовнішньої пам'яті даних, або записування даних у порт D0 – D7
T1	39	Вхідний сигнал, що опитується командами умовного переходу JT1 та JNT1. Також може використовуватися в якості входу лічильника зовнішніх подій
T0	1	Вхідний сигнал, що опитується командами умовного переходу JT0 та JNT0. Також може працювати як вихід сигналу з частотою, що дорівнює 1/3 частоти тактового сигналу
D0 – D7 (BUS)	12-19	Порт 0. Восьмибітний двонаправлений порт вводу-виводу інформації. Дані, що передаються, стробуються сигналами записування (W) і читання (R)

1	2	3
P2.0 – P2.7	21-24, 35-38	Порт 2. Восьмибітний квазідвонаправлений порт вводу-виводу 2, кожен розряд якого може працювати як вхід або вихід незалежно від інших. Біти P2.3-P2.0 під час читання із зовнішньої пам'яті програм змінюють своє значення, на 4 старших розряди адреси коду команди, що виконується на весь час вибірки коду команди. Після завершення зчитування коду дані відновлюються із внутрішніх тригерів порту, крім того, вони можуть бути зчитані при виконанні відповідних команд
P1.0 – P1.7	27-34	Порт 1. Восьмибітний квазідвонаправлений порт вводу-виводу 1, кожен розряд якого може працювати як вхід або вихід незалежно від інших

3. Основні функціональні блоки ОМЕОМ

Мікроконтролер K1816BE48 містить (рис. 49): центральний процесор ЦП (8 біт), пам'ять програм РПЗУ ємністю 1 кбайт; пам'ять даних ОЗП ємністю 64 байти, багатоканальний інтерфейс вводу-виводу P0, P1, P2 (27 ліній), програмований 8-бітний таймер-лічильник ТС, регістр адреси РА, пристрій керування та синхронізації, дешифратор команд ДК, регістр команд РК. Передбачено розширення пам'яті РПЗП до 4 кбайтів, ОЗП до 256 байтів, а також збільшення кількості ліній вводу-виводу за допомогою під'єднання зовнішніх ПЗП, ОЗП та інтерфейсів вводу-виводу серії K580 та інших.

Структура мікросхеми зображена на рис. 47. Розглянемо призначення її основних блоків.

Блок процесора складається з 7 підблоків:

1. Акумулятор (А) використовується як 8-розрядний регістр операнда та регістр результату.

2. Регістр вибірки та зберігання (РВЗ) призначений для приймання та тимчасового зберігання другого байта команди, а також для збереження проміжних даних зчитаних із портів вводу-виводу чи комірок пам'яті.

3. Арифметико-логічний пристрій (АЛП), призначений для формування результатів арифметичних і логічних операцій, які виконує центральний процесор.

RETI. Біт керування виходом T0CLK може лише бути встановленим у 1. Одиничне значення біта під'єднує до виходу T0 тактовий сигнал. Скидання біта здійснюється у момент скидання мікросхеми.

7. Регістр слова стана процесора (РССП) призначений для збереження вказівника стека та ознак результату останньої арифметичної або логічної операції. Формат регістра слова стана програми зображено на рис. 48.



Рис. 48. Формат регістра слова стана процесора

Біт перенесення C використовується для фіксації перенесення при виконанні команд додавання та циклічного зсуву. Біт додаткового перенесення AC указує на наявність перенесення із третього розряду акумулятора у четвертий при виконанні арифметичних команд. Біт використовується при роботі з двійково-десятковими числами. Біт F0 носить назву ознаки користувача, він використовується для збереження ознаки, що встановлюється та скидається програмним шляхом, стан розряду може бути перевірений спеціальною командою. Біт BS носить назву "селектор банку регістрів" та призначений для визначення активного банку регістрів. Біти S2, S1, S0 створюють вказівник стека та призначені для визначення зміщення вершини стека від початкової адреси. Адреса вершини стека обчислюється за формулою $S2S1S0 \times 2 + 8$. Старші чотири розряди РССП можуть бути збережені у стеку в процесі викликів підпрограм чи підтвердження переривання.

Пристрій керування і синхронізації (ПКС), призначений для керування і синхронізації роботи інших блоків мікро-ЕОМ залежно від виконуваного коду команди та стана ліній мікросхеми. У ньому виділяють два підблоки: регістр команд (РК), котрий містить код виконуваної команди та дешифратор команд (ДК), призначений для перетворення коду команди в сигнали керування.

Блок пам'яті даних складається з ОЗП, дешифратора і регістра адреси

(РА). Дешифратор призначений для визначення позиції комірки ОЗП, на основі адреси збереженої в РА. Оперативний запам'ятовуючий пристрій (ОЗП) використовується лише як пам'ять даних. Він служить для записування, зберігання та зчитування даних у процесі роботи програми інформації. Загальний об'єм ОЗП залежить від типу мікросхеми.

В ОЗП виділені два банки регістрів загального призначення, та восьмирівневий 16-розрядний стек. Решта комірок пам'яті використовується у якості ОЗП загального призначення для збереження даних. Спрощена структура оперативної пам'яті зображена рис. 49.

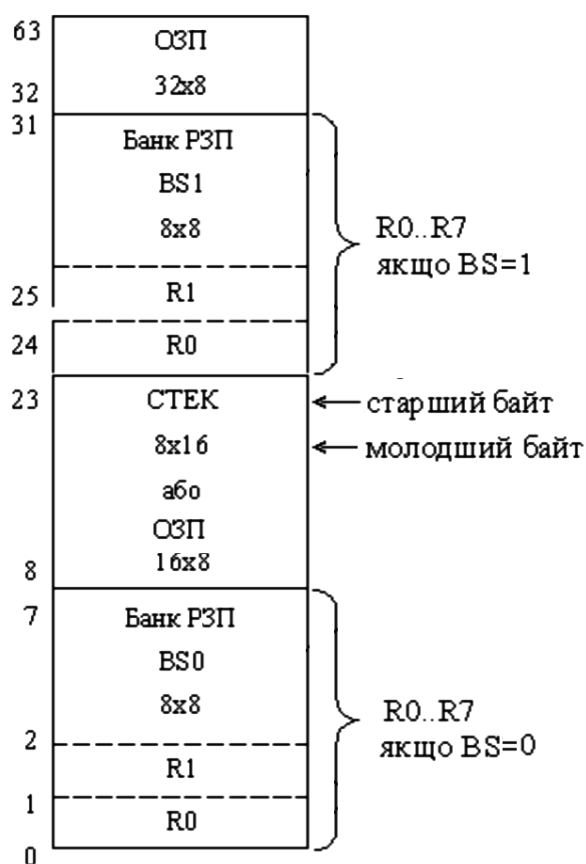


Рис. 49. Структура ОЗП КР1816ВЕ48

Банки регістрів загального призначення (РЗП) розміщені у комірках з адресами 0÷7 та 24÷31 (18h ÷ 1Fh). Кожен банк має по вісім 8-розрядних регістрів. Регістри кожного банку призначені для тимчасового зберігання даних. Регістри загального призначення використовувати простіше ніж звичайні комірки пам'яті. За допомогою спеціальної команди BS0 чи BS1 можна легко змінити активний банк регістрів. Фактично, банк регістрів

подібний на сегмент стана задачі у потужних процесорах.

Восьмирівневий 16-розрядний стек (адреси $8 \div 23$) використовується для збереження адреси повернення з підпрограм. У разі необхідності його ж можна використовувати у якості стека даних, якщо програміст буде сам змінювати вказівник стека. Обсяг стека обмежений – у нього може бути поміщено лише 8 двобайтових слів, а переповнення стека ні як не контролюється, тому при використанні стека слід бути обережним. У разі можливості переповнення стека варто контролювати його стан за допомогою розрядів $S2 \div S0$ у РССП. ОЗП розміщене у старших адресах може використовуватися лише як ОЗП даних. Блок РПЗП призначений для збереження та вибірки кодів команд, що виконуються мікро-ЕОМ. Він складається з: репрограмованого постійного запам'ятовуючого пристрою (РПЗП) ємністю 1 чи 2 кбайти, котрий призначений для збереження програм; дешифратора, для дешифрування номера комірки РПЗП та лічильника команд старшого (ЛК11-ЛК8) та молодшого байта адреси (ЛК7-ЛК0).

Блок вводу-виводу призначений для вводу і виводу інформації з мікро-ЕОМ через порти $P0 - P2$. На схемі цьому блоку відповідають підблоки з однойменними назвами. Останнім блоком, що розглядається, є блок лічильника-таймера (ЛТ), він являє собою 8-розрядний лічильник, якій може працювати в режимі таймера. ЛТ може генерувати переривання при переповненні.

3.1. Порти

Багатоканальний інтерфейс вводу-виводу призначений для обміну інформацією ОМЕОМ з периферійними пристроями. ОМЕОМ має 27 ліній вводу-виводу, 24 з яких об'єднані в три 8-розрядних канали $P2, P1, P0$. Канали $P1, P2$ у процесі роботи мають можливість фіксації даних. Ці дані є на виводах каналу і можуть бути змінені тільки новою командою. $P1, P2$ можуть працювати на ввід або на роботу з двонаправленою лінією передачі.

Порти $P1$ і $P2$ є керованими буферними регістрами. Вихід кожного біта має вихід з відкритим колектором та внутрішнім резистором, що підтягує лінію

до рівня напруги живлення. Резистор шунтується додатковим транзистором. Він вмикається лише на короткий час у момент перемикання лінії у стан логічної 1 для прискорення перемикання лінії.

У момент виводу байт фіксується у буферному регістрі порту. Стан кожної лінії може бути прочитаний протягом усього часу присутності байта на входах порту. Орієнтація МК на застосування у пристроях керування об'єктами посприяла появі команд, що дозволяють виконувати операції вводу-виводу над окремими бітами та довільними їх комбінаціями. Команди логічного АБО та логічного І над вмістом порта P1 та P2 виконуються над записаними у порти даними та дозволяють змінювати лише потрібні біти без додаткових операцій читання.

На відміну від операцій логічного І та АБО над вмістом порта, будь-яка команда читання порта зчитує стан вихідних ліній. Зчитані з порта дані можуть не відповідати даним, що були записані у нього. З іншого боку, якщо лінія повинна працювати як вхід, необхідно у відповідний розряд порта попередньо записати 1, у противному випадку буде зчитуватись 0, а лінія буде закорочена на землю.

Лінії P2.3÷P2.0 використовуються як виходи для передавання адреси під час читання із зовнішньої пам'яті програм та як двонаправлені лінії для роботи з розширювачем вводу-вивода. Вони мають внутрішній комутатор, що відмикає квазідвонаправлений вихід і підключає звичайний МОН вихід на час видавання адреси та роботи з розширювачем, що збільшує швидкодію ліній. Після завершення зчитування коду або роботи з розширювачем комутатор знову вмикає квазідвонаправлений вихід. Тому не має необхідності відновлювати стан порта P2 після кожної команди роботи з зовнішньою пам'яттю команд.

Кожна лінія порта BUS (P0) має звичайний тристабільний вихід. Напрямок передачі задається типом останньої виконаної портом команди: якщо останньою була команда вивода – порт працює на виведення даних, якщо команда введення – на введення даних, якщо ж останньою була команда обміну

з зовнішньою пам'яттю даних – лінії порта переводяться у третій стан. Скидання мікросхеми перемикає всі лінії BUS на введення даних. Порт BUS може виконувати всі функції портів P1 і P2, але, на відміну від них, він не може програмувати окремі лінії на ввід чи вивід. Усі вісім ліній порту BUS мають одночасно бути вхідними, або вихідними. За командою MOVX порт BUS використовується в якості двонаправленого синхронного каналу для доступу до зовнішньої пам'яті, на відміну від команд INS та OUTL, виконання команди MOVX завжди супроводжується виведенням адреси комірки зовнішньої пам'яті даних.

Стан ліній T0, T1, INT можна перевірити командами умовного переходу. Лінія T1 може працювати як джерело імпульсів для лічильника-таймера, лінія INT може працювати входом переривань, а лінія T0 як вихід тактового сигналу.

3.2. Лічильник-таймер

Внутрішній 8-бітний двійковий додаючий лічильник використовують для формування часових інтервалів та визначення кількості зовнішніх подій. Вміст лічильника-таймера (T) можна прочитати (командою MOV A, T) або змінити (MOV T, A) не залежно від режиму роботи. Немає необхідності зупиняти лічильник на час зчитування або записування даних.

Спеціальні команди налаштовують та запускають лічильник-таймер у режимі таймера або у режимі лічильника подій. Зупинити роботу лічильника-таймера можна або командою STOP TCNT, або сигналом системного скиду CLR. У момент переходу від максимального значення (FFh) у стан 00h, встановлюється в 1 ознака переповнення таймера TF, що може викликати переривання таймера, якщо воно дозволене командою EN TCNTI. Переривання від лічильника-таймера може бути заборонене командою DIS TCNTI. Ознака переповнення може бути опитана командою умовного переходу JTF. Команда JTF та виклик переривання від таймера скидають ознаку TF.

У режимі таймера на вхід лічильника-таймера подається сигнал ALE через подільник на 32. Сигнал ALE має частоту 1/15 частоти сигналу

резонатора, тому частота на вході лічильника рівна $1/480$ частоти резонатора. Лічильник-таймер може визначити максимальну тривалість періода рівну 122880 періодам частоти резонатора. У випадку необхідності, у ОЗП може бути організований програмний лічильника, розширюючий розрядність лічильника до 16 біт.

У режимі лічильника подій лічильник збільшує свій стан на 1, у момент переходу T1 зі стану 1 у стан 0. Мінімальна тривалість періоду дорівнює 3 машині цикли. Мінімальна тривалість одиничного сигналу на вході T1 складає 0.5 мкс.

3.3. Переривання

Контролер має 2 джерела переривань: переривання за переповненням лічильника-таймера та переривання за станом лінії INT. З кожним перериванням зв'язана початкова адреса процедури обробки.

Лінія переривання від зовнішнього джерела INT перевіряється кожний цикл у момент високого рівня сигналу ALE. У випадку наявності на лінії INT логічного нуля виконується команда CALL 3h. Команда здійснює перехід на адресу 3h, за котрою знаходиться команда JMP на початок підпрограми опрацювання переривання. До входу INT мікроконтролера може бути підключені декілька джерел переривань. Після розпізнавання переривання всі наступні запити переривання ігноруються до тих пір, поки по команді повернення RETR знову буде дозволена робота логіки переривань. Режим переривань може бути заборонений або дозволений у програмі за командами DIS I та EN I. Сигнал INT повинен бути знятим перед закінченням підпрограми обслуговування, тобто до виконання команди RETR.

У разі необхідності можна створити дворівневу систему переривань. Для цього потрібно дозволити переривання від таймера, завантажити в нього число FFh і перевести його у режим підрахунку кількості зовнішніх подій. Зміна на вході T1 з стану 1 в стан 0 приведе до переривання за вектором 7. У випадку одночасного запиту переривання від INT та таймера, буде обслуговуватись

переривання від INT. На початку підпрограми обслуговування переривання старший біт лічильника команд примусово встановлюється у нуль. Відповідно, вся процедура обробки переривання повинна бути розміщена в банку пам'яті 0.

4. Система команд МК48

4.1. Загальні відомості про систему команд

Система команд МК48 містить у собі 96 основних команд і орієнтована на реалізацію процедур керування. Код усіх команд складається із одного чи двох байт, 70% команд – однобайтні. Час виконання команд – один чи два машинні цикли, що складає 2,5 чи 5 мкс при тактовій частоті 6 МГц.

Більшість команд виконується за один машинний цикл. За два машинні цикли виконуються команди із безпосереднім операндом, вводу-виводу і передавання керування. МК48 оперує з командами, котрі за форматом кода команди розділяються на чотири типи. Формат кожного з кодів команди зображений на рис. 50. В усіх типах команд перший байт повністю визначає характер та операнди команди. У командах першого типу інформації першого байта достатньо для визначення операндів команди та операції, що виконується. У командах другого типу другий байт містить константу, що використовується у якості одного з операндів. Команди третього типу задають у другому та частково у першому байті адресу переходу, а команди четвертого типу адресу операнда задають у межах сторінки розміром 256 байтів.

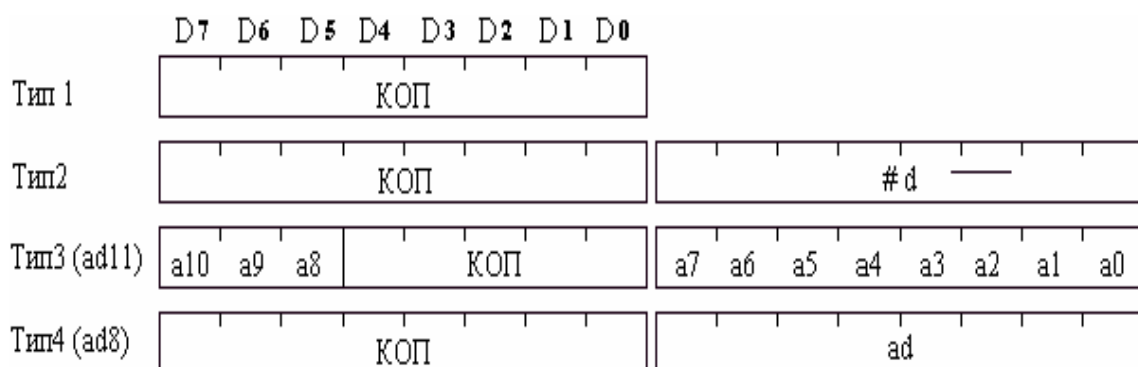


Рис. 50. Типи команд МК48

Переваги системи команд МК48: ефективний ввід-вивід, можливість керувати окремими бітами портів; наявність команд розгалуження за значенням

бітів; обробка як двійкових, так і двійково-десяткових чисел.

4.2. Класифікація команд МК48

У МК48 використовуються чотири способи адресації: пряма, безпосередня, непряма та неявна. Усі команди можна розбити на п'ять груп за функціональною ознакою: команди пересилання даних, арифметичні операції, логічних операцій, передачі керування і керування режимами роботи МК.

Нижче коротко описані команди МК48, згруповані за функціональною ознакою. При цьому використані коди мови асемблер МК48, а операції, виконувані за командами, описуються мікрооператорами із використанням символічних імен та скорочень.

4.3. Способи адресації даних

Для звертання до даних використовують чотири способи адресації:

- ◇ Пряма, коли адреса операнда міститься у тілі самої команди, наприклад номер регістра може бути заданий у трьох молодших бітах коду операції: `MOV A, Rn`.
- ◇ Безпосередня, коли сам 8-бітний операнд (константа) розташовується безпосередньо в тілі команди (у другому байті), наприклад: `MOV A, #05 (A = 5)`.
- ◇ Непряма, при якій адреса одного з операндів розташовується у регістрі A, R0 чи R1, наприклад: `MOV A, @R0 (A = IRAM[R0])`.
- ◇ Неявна, при якій у коді операції розміщена вказівка на один з операндів. Найчастіше таким операндом є акумулятор, як, наприклад, у команді `MOV A, PSW (A = PSW)`.

4.4. Модифікація РССП та ознак користувача

При виконанні команд можуть змінюватись значення ознак користувача та окремих бітів РССП. Усі команди, через які модифікуються ознаки наведені у таблиці 27. Зауважимо, що логічні команди не змінюють регістр ознак, проте МК-48 має команди перевірки результатів виконання логічних операцій, без використання регістра РССП.

Таблиця 27. Команди модифікації ознак

Команди	Ознаки	Опис команди
1	2	3
ADD	C, AC	Команда додавання до акумулятора
ADDC	C, AC	Команда додавання до акумулятора з урахуванням перенесення
DA A	C, AC	Команда десяткової корекції після додавання
CLR C	C = 0	Скидання ознаки перенесення
CPL C	C	Інверсія ознаки перенесення
CLR F0	F0	Скидання ознаки користувача F0
CLR F1	F1	Скидання ознаки користувача F1
CPL F0	F0	Інверсія ознаки користувача F0
CPL F1	F1	Інверсія ознаки користувача F1
JTF	TF = 0	Перехід, якщо біт TF рівний 1. Команда скидає біт переповнення таймера TF
MOV PSW, A	C, AC, F0, BS	Записування даних у РСЦП
RETR	C, AC, F0, BS, GEI	Повернення з підпрограми або переривання
RLC	C	Лівий циклічний зсув бітів через біт C
RRC	C	Правий циклічний зсув бітів через ознаку перенесення
SEL MB0 SEL MB1	MB	Вибір банків пам'яті програм. Команда задає нульовий чи перший банк пам'яті програм
SEL RB0 SEL RB1	BS	Вибір банків регістрів. Команда активізує нульовий або перший банк регістрів загального призначення

4.5. Група команд пересилання даних

Ця група складається з 25 команд. Усі команди (крім MOV PSW, A) не впливають на ознаки. Команди пересилання даних усередині МК виконуються за один машинний цикл, обмін із зовнішньою пам'яттю і портами вимагає двох машинних циклів.

На рис. 51 зображений граф можливих пересилань, що ілюструє структуру інформаційних зв'язків МК48. Можна виділити дев'ять типів операндів, між якими виконується інформаційний обмін. Операнди, що беруть участь в операціях пересилання, розрізняються за місцем розташування і способом адресації.

До операндів належать: акумулятор (A), РСЦП (PSW), таймер (T),

реєстри загального призначення (Rn), безпосередній операнд (#Data), зовнішня пам'ять даних (ЗПД), внутрішня пам'ять даних (ВПД), порти (P0-P7) та пам'ять програм (ROM).

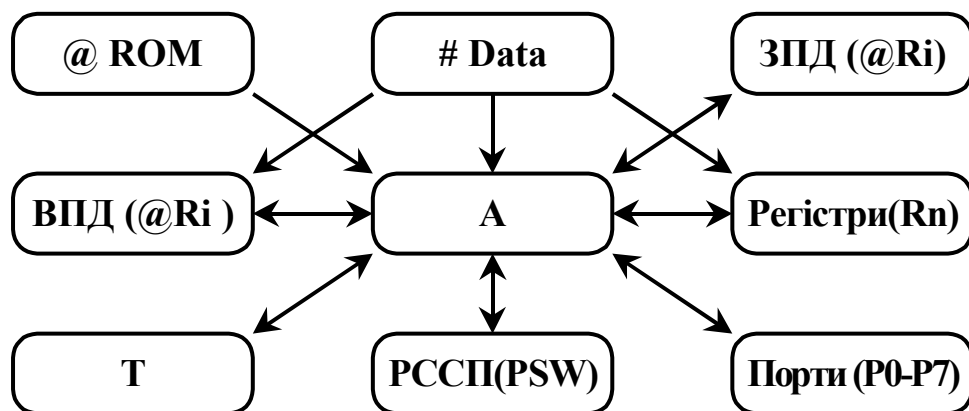


Рис. 51. Граф пересилання даних

З рисунка випливає, що акумулятор використовується як "поштова скринька", через яку інші пристрої можуть обмінюватися даними. До пам'яті програм існує тільки односторонній доступ для читання.

Більшість команд пересилає 8-бітні (байтові) операнди. Існують кілька команд, що оперують з 4-бітними операндами (тетрадами). Команди пересилання тетрад використовують при звертанні до 4-бітних портів зовнішньої схеми розширювача вводу-виводу (P4÷P7).

Формат, дія та назва кожної команди пересилання даних наведені у таблиці 28. У таблиці використані позначення: Т – тип команди (за рис. 50), Б – кількість байтів у коді команди, Ц – тривалість виконання команди у машинних циклах.

Таблиця 28 Команди пересилання даних

Позначення	Код	Т	Б	Ц	Назва та дія команди
1	2	3	4	5	6
MOV A, Rn	11111rrr	1	1	1	Пересилання реєстра в акумулятор: A = Rn
MOV A, @Ri	111000i	1	1	1	Пересилання байта з ВПД в акумулятор, адреса комірки внутрішньої пам'яті даних знаходиться у реєстрі R0 (R1): A = IRAM[Ri]
MOV A, #d	00100011 d _{7÷0}	2	2	2	Пересилання безпосереднього операнда в акумулятор: A = #d
MOV Rn, A	10101rrr	1	1	1	Пересилання акумулятора в реєстр: Rn = A

Продовження таблиці 28

1	2	3	4	5	6
MOV Rn, #d	10111rrr d _{7÷0}	2	2	2	Пересилання безпосереднього операнда в регістр: Rn = #d
MOV @Ri, A	1010000i	1	1	1	Пересилання акумулятора у внутрішню пам'ять даних, адреса комірки пам'яті знаходиться у регістрі R0 чи R1: IRAM[Ri] = A
MOV @Ri, #d	1011000i d _{7÷0}	2	2	2	Пересилання безпосереднього операнда у внутрішню пам'ять даних, адреса комірки знаходиться у регістрі R0 чи R1: IRAM[Ri] = #d
MOV A, PSW	11000111	1	1	1	Пересилання слова стана програми в акумулятор: A = PSW
MOV PSW, A	11010111	1	1	1	Пересилання акумулятора у ПССП: PSW = A
MOV A, T	10000100	1	1	1	Пересилання вмісту лічильника-таймера в акумулятор: A = T
MOV T, A	01100010	1	1	1	Пересилання акумулятора в лічильник-таймер: T = A
MOVX A,@Ri	1000000i	1	1	2	Пересилання байта із зовнішньої пам'яті даних в акумулятор, адреса комірки пам'яті даних знаходиться у регістрі R0 чи R1: A = ERAM[Ri]
MOVX @Ri,A	1001000i	1	1	2	Пересилання акумулятора у зовнішню пам'ять даних, адреса комірки пам'яті знаходиться у регістрі R0 чи R1: ERAM[Ri] = A
MOVP A,@A	10100011	1	1	2	Пересилання байта з поточної сторінки програмної пам'яті в акумулятор: A = ROM[PC _{11÷8} :A]
MOVP3 A,@A	11100011	1	1	2	Пересилання байта з третьої сторінки програмної пам'яті в акумулятор: A = ROM[0011:A]
XCH A,Rn	00101rrr	1	1	1	Обмін регістра з акумулятором: A ↔ Rn
XCH A,@Ri	0010000i	1	1	1	Обмін акумулятора з внутрішньою пам'яттю даних, адреса комірки пам'яті даних знаходиться у регістрі R0 чи R1: A ↔ IRAM[Ri]
XCHD A,@Ri	0011000i	1	1	1	Обмін молодших тетрад акумулятора та з внутрішньою пам'яттю даних, адреса комірки пам'яті знаходиться у регістрі R0 чи R1: A _{3...0} ↔ IRAM[Ri] _{3...0}
IN A,Pp	000010pp	1	1	2	Пересилання даних з портів вводу-виводу Pp (p =1,2) в акумулятор: A = Pp

1	2	3	4	5	6
INS A,BUS	00001000	1	1	2	Стробоване введення даних з P0: $A = BUS$
SWAP A	01000111	1	1	1	Обмін тетрад акумулятора: $A_{3...0} \leftrightarrow A_{7...4}$
OUTL P _p ,A	001110pp	1	1	2	Пересилання A у порти P1 та P2 ($P_p p = 1, 2$): $P_p = A$
OUTL BUS, A	00000010	1	1	2	Стробований вивід даних з A в порт BUS: $BUS = A$
MOVD A,P _p	000011pp	1	1	2	Ввід тетради з порту P _p ($p = 4 - 7$) схеми розширення: $A_{3...0} = P_p$
MOVD P _p ,A	001111pp	1	1	2	Вивід тетради в порт P _p ($p = 4 - 7$) схеми розширення: $P_p = A_{3...0}$

У МК48 можливо передавати дані у режимі пересилання (чи завантаження) та у режимі обміну. При пересиланні дані можна передавати від джерела до приймача. При цьому джерело не змінює свого вмісту. Команди обміну передають дані у двох напрямках та змінюють значення обох операндів.

4.6. Група команд арифметичні операції

Ця група складається з 12 команд і дозволяє виконувати такі операції над 8-бітними цілими двійковими числами без знака: двійкове додавання (ADD), двійкове додавання з урахуванням перенесення (ADDC), десяткова корекція (DAA), інкрементування (INC) і декрементування (DEC) регістрів МК48. Опис команд наведений у таблиці 29.

Таблиця 29. Група команд арифметичних операцій

Позначення	Код	T [*])	B [*])	C [*])	Назва та дія команди
1	2	3	4	5	6
ADD A,R _n	01101rrr	1	1	1	Додавання регістра R0 – R7 до акумулятора: $A = A + R_n$
ADD A,@R _i	0110000i	1	1	1	Додавання до акумулятора байта із внутрішньої пам'яті даних. Адреса комірки пам'яті знаходиться у регістрі R0 чи R1: $A = A + IRAM[R_i]$
ADD A,#d	00000011 d _{7÷0}	2	2	2	Додавання до акумулятора константи: $A = A + \#d$
ADDC A,R _n	01111rrr	1	1	1	Додавання до акумулятора регістра R0 – R7 та ознаки перенесення: $A = A + R_n + C$
ADDC A,@d	00010011 d _{7÷0}	2	2	2	Додавання до акумулятора константи та ознаки перенесення: $A = A + \#d$

Закінчення таблиці 29

1	2	3	4	5	6
ADDC A,@Ri	0111000i	1	1	1	Додавання до акумулятора байта із внутрішньої пам'яті даних та ознаки перенесення. Адреса комірки пам'яті знаходиться у регістрі R0 чи R1: $A = A + \text{IRAM}[\text{Ri}] + C$
DAA	01010111	1	1	1	Десяткова корекція акумулятора. Команда виправляє результат додавання за типовими правилами
INC A	00010111	1	1	1	Інкремент акумулятора: $A = A + 1$
INC Rn	00011rrr	1	1	1	Інкремент регістра: $Rn = Rn + 1$
INC @Ri	0001000i	1	1	1	Інкремент байта у пам'яті. Адреса комірки пам'яті знаходиться у регістрі R0 чи R1: $\text{IRAM}[\text{Ri}] = \text{IRAM}[\text{Ri}] + 1$
DEC A	00000111	1	1	1	Декремент акумулятора: $A = A - 1$
DEC Rn	11001rrr	1	1	1	Декремент регістра R0 – R7: $Rn = Rn - 1$
*) У таблиці використані позначення: Т – тип команди (за рис. 50), Б – кількість байтів у коді команди, Ц – тривалість виконання команди у машинних циклах					

Команда додавання ADD додає до акумулятора інший операнд. Вміст акумулятора А можна просумувати з регістром загального призначення, з константою, або коміркою внутрішньої пам'яті даних. Старший (дев'ятий) біт результату фіксується у біті С регістра РССП. Команда додавання з урахуванням перенесення ADDC працює аналогічно команді додавання ADD, але до суми додає попереднє значення ознаки перенесення. Команда призначена для сумування багатобайтових чисел.

У системі команд відсутні команди віднімання, множення, ділення. У процесі написання програми команди віднімання необхідно замінювати на команди додавання з числом записаним у додатковому коді. Операції множення та ділення слід оформляти підпрограмами.

4.7. Група команд роботи з бітами

Група складається із 27 команд і дозволяє виконувати такі операції: логічне І, логічне АБО, виключне АБО, інверсію, скидання і зсув. Дві команди (скидання й інверсія) дозволяють виконувати операції над бітами. Формат та характеристики команд наведені у таблиці 30.

Таблиця 30. Група команд роботи з бітами

Позначення	Код	T [*])	B [*])	Ц [*])	Назва та дія команди
1	2	3	4	5	6
ANL A,Rn	01011rrr	1	1	1	Логічне І акумулятора й регістра R0 ÷ R7: $A = A \wedge Rn$
ANL A,@Ri	0101000i	1	1	1	Логічне І байта акумулятора та комірки внутрішньої пам'яті даних. Адреса комірки пам'яті знаходиться у регістрі R0 чи R1: $A = IRAM[Ri] \wedge A$
ANL A,#d	01010011 d _{7÷0}	2	2	2	Логічне І константи та акумулятора: $A = A \wedge \#d$
ORL A,Rn	01001rrr	1	1	1	Логічне АБО акумулятора та регістра R0 ÷ R7: $A = A \vee Rn$
ORL A,@Ri	0100000i	1	1	1	Логічне АБО акумулятора та комірки внутрішньої пам'яті даних. Адреса комірки пам'яті знаходиться у регістрі R0 чи R1: $A = IRAM[Ri] \vee A$
ORL A,#d	01000011 d _{7÷0}	2	2	2	Логічне АБО акумулятора та константи: $A = A \vee \#d$
XRL A,Rn	11011rrr	1	1	1	Виключне АБО акумулятора й регістра R0 ÷ R7: $A = A \oplus Rn$
XRL A,@Ri	1101100i	1	1	1	Виключне АБО акумулятора та комірки внутрішньої пам'яті даних. Адреса комірки пам'яті знаходиться у регістрі R0 чи R1: $A = A \oplus IRAM[Ri]$
XRL A,#d	11010011 d _{7÷0}	2	2	2	Виключне АБО константи й акумулятора: $A = A \oplus \#d$
CPA	00110111	1	1	1	Інверсія акумулятора: $A = \overline{A}$
CLR A	00100111	1	1	1	Скидання акумулятора: $A = 0$
RL A	11100111	1	1	1	Циклічний зсув вліво вмісту акумулятора: $A_{n+1} = A_n, n = 0 \div 6, A_0 = A_7$
RLC A	11110111	1	1	1	Зсув вліво акумулятора через ознаку перенесення: $A_{n+1} = A_n, n = 0 \dots 6, A_0 = C, C = A_7$
RR A	01110111	1	1	1	Циклічний зсув вправо акумулятора: $A_n = A_{n+1}, n = 0 \div 6, A_7 = A_0$
RRC A	01100111	1	1	1	Зсув вправо акумулятора через перенесення: $A_n = A_{n+1}, n = 0 \div 6, A_7 = C; C = A_0$
ANL Pp,#d	100110pp d _{7÷0}	2	2	2	Логічне І константи та даних з порта Pp: $Pp = Pp \wedge \#d, p = 1, 2$
ANL BUS,#d	10011000 d _{7÷0}	2	2	2	Логічне І константи та даних з порта BUS: $BUS = BUS \wedge \#d$
ANLD Pp,A	100111pp	1	1	2	Логічне І акумулятора та даних з порта Pp (p = 7 ÷ 4): $Pp = Pp \wedge A_{3-0}$

Закінчення таблиці 30

1	2	3	4	5	6
ORL BUS,#d	10001000 d _{7÷0}	2	2	2	Логічне АБО константи та даних BUS: BUS = BUS ∨ #d
ORLD Pp,A	100011pp	1	1	2	Логічне АБО акумулятора і порту Pp (p = 7 ÷ 4): Pp = Pp ∨ A _{3...0}
CLR C	10010111	1	1	1	Скидання ознаки перенесення C = 0
CLR F0	10000101	1	1	1	Скидання ознаки користувача F0 (F0 = 0)
CLR F1	10100101	1	1	1	Скидання ознаки користувача F1 (F1 = 0)
CPL C	10100111	1	1	1	Інверсія ознаки перенесення C = $\overline{C}$
CPL F0	10010101	1	1	1	Інверсія ознаки користувача F0 (F0 = $\overline{F0}$)
CPL F1	10110101	1	1	1	Інверсія ознаки користувача F1 (F1 = $\overline{F1}$)
*) У таблиці використані позначення: Т – тип команди (за рис. 50), Б – кількість байтів у коді команди, Ц – тривалість виконання команди у машинних циклах					

У командах обробки бітів широко використовується неявна адресація акумулятора як джерело операції і місце фіксації результату. Другим операндом у командах може бути регістр, константа або комірка РПД.

Команди, які оперують з портами, дозволяють ефективно керувати значеннями окремих бітів при вводі-виводі інформації. Команди виконання логічних операцій, котрі працюють з портами P4÷P7, виконує розширювач вводу-вивода K580BB43: мікроконтролер видає через лінії P2.3 ÷ P2.0 код команди та необхідні дані, а дії над даними виконує K580BB43.

4.8. Група команд передавання керування

Групу утворюють 19 команд передавання керування, з них дві команди безумовного переходу, 14 команд умовного переходу, команда виклику підпрограм і дві команди повернення з підпрограм.

У таблиці 31 наведено формат та робота кожної з команд. Команди умовних переходів здійснюють перехід лише у випадку виконання певної умови. Якщо ж умова не справджується здійснюється перехід до наступної команди. Команди CALL, RET та RETR виконують дії зі стеком.

У більшості команд прямо вказана адреса переходу. У тілі команди при цьому зберігаються 8 (ad_{7÷0}) чи 11 (ad_{10÷0}) бітів адреси переходу. Команда JMP

дозволяє передати керування у будь-яке місце 2048-байтного банку пам'яті програм (ПП). Номер банку ПП визначається ознакою DBF, значення якого копіюється у старший біт лічильника команд (PC11) при виконанні команди JMP чи CALL. Для переходу з нульового банку ПП у перший недостатньо тільки встановити ознаку DBF, необхідно також виконати команду JMP або CALL, що змінить значення старшого біта лічильника команд.

Таблиця 31. Команди передавання керування

Позначення	Код	T ^{*)}	B ^{*)}	C ^{*)}	Назва та дія команди
1	2	3	4	5	6
JMP ad11	a ₁₀ a ₉ a ₈ 00100 a _{7÷0}	3	2	2	Безумовний перехід: PC ₁₁ = DBF, PC _{10÷0} = a _{10÷0}
JMPP @A	10110011	1	1	2	Непрямий перехід у середині сторінки: PC _{7÷0} = A
DJNZ Rn, ad	11101rrr ad _{7÷0}	4	2	2	Декремент регістра та перехід, якщо результат ненульовий. Здійснюється перехід на адресу задану другим байтом у середині сторінки пам'яті програм
JC ad	11110110 ad _{7÷0}	4	2	2	Умовний перехід, за перенесенням. Якщо ознака C = 1, то PC _{7÷0} = ad _{7÷0}
JNC ad	11100110 ad _{7÷0}	4	2	2	Умовний перехід, у випадку відсутності перенесення. Якщо C=0, то PC _{7÷0} = ad _{7÷0}
JZ ad	11000110 ad _{7÷0}	4	2	2	Умовний перехід, за нульовим значенням акумулятора. Якщо A = 0, то PC _{7÷0} = ad _{7÷0}
JNZ ad	10010110 ad _{7÷0}	4	2	2	Перехід, якщо значення акумулятора ненульове. Якщо A ≠ 0, PC _{7÷0} = ad _{7÷0}
JT0 ad	00110110 ad _{7÷0}	4	2	2	Умовний перехід за високим рівнем на вході T0. Якщо T0 = 1, то PC _{7÷0} = ad _{7÷0}
JNT0 ad	00100110 ad _{7÷0}	4	2	2	Умовний перехід за низьким рівнем на вході T0. Якщо T0 = 0, то PC _{7÷0} = ad _{7÷0}
JT1 ad	01010110 ad _{7÷0}	4	2	2	Умовний перехід за високим рівнем на вході T1. Якщо T1 = 1, то PC _{7÷0} = ad _{7÷0}
JNT1 ad	01000110 ad _{7÷0}	4	2	2	Умовний перехід за низьким рівнем на вході T1. Якщо T1 = 0, то PC _{7÷0} = ad _{7÷0}
JF0 ad	10110110 ad _{7÷0}	4	2	2	Умовний перехід за станом ознаки F0. Якщо F0 = 1, то PC _{7÷0} = ad _{7÷0}
JF1 ad	01110110 ad _{7÷0}	4	2	2	Умовний перехід за станом ознаки F1. Якщо F1 = 1, то PC _{7÷0} = ad _{7÷0}

Закінчення таблиці 31

1	2	3	4	5	6
JTF ad	00010110 ad _{7÷0}	4	2	2	Умовний перехід за станом таймера. Якщо таймер з моменту попереднього виконання команди переповнювався (ознака TF = 1), то PC _{7÷0} = ad _{7÷0} , а TF обнулюється
JNI ad	10000110 ad _{7÷0}	4	2	2	Умовний перехід за станом лінії INT. Якщо на виводі INT логічний 0, то PC _{7÷0} = ad _{7÷0}
JBb ad	b ₂ b ₁ b ₀ 10010 ad _{7÷0}	4	2	2	Умовний перехід, у випадку рівності 1 бітів акумулятора. Якщо біт регістра A, номер котрого заданий кодом b _{2÷0} дорівнює одиниці, то PC _{7÷0} = ad _{7÷0}
CALL ad11	a ₁₀ a ₉ a ₈ 10100 ad _{7÷0}	3	2	2	Виклик підпрограми. Адреса команди, що йде за командою виклика записується у стек: Stack[S] = (PSW _{7÷4} , PC+2), S = S + 1, PC ₁₁ = DBF, PC _{10÷0} = ad _{10÷0}
RET	10000011	1	1	2	Повернення з підпрограми зі збереженням слова стану: S = S – 1, PC = Stack[S]
RETR	10010011	1	1	2	Повернення з підпрограми з відновленням слова стану: S = S – 1, (PSW _{7÷4} , PC) = Stack[S]
*) У таблиці використані позначення: Т – тип команди (за рис. 50), Б – кількість байтів у коді команди, Ц – тривалість виконання команди у машинних циклах					

У більшості команд напряду вказана адреса переходу. У тілі команди при цьому зберігаються 8 (ad_{7÷0}) чи 11 (ad_{10÷0}) бітів адреси переходу. Команда JMP дозволяє передати керування у будь-яке місце 2048-байтного банку пам'яті програм (ПП). Номер банку ПП визначається ознакою DBF, значення якого копіюється у старший біт лічильника команд (PC₁₁) при виконанні команди JMP чи CALL. Для переходу з нульового банку ПП у перший недостатньо тільки встановити ознаку DBF, необхідно також виконати команду JMP або CALL, що змінить значення старшого біта лічильника команд.

Усі інші команди (крім команд повернення) використовують лише вісім молодших бітів адреси. Старші чотири розряди лишаються незмінними. Отже

перехід здійснюється лише у межах однієї сторінки пам'яті програм. Якщо команда короткого переходу знаходиться на границі двох сторінок (тобто перший байт команди на одній сторінці, а другий – на наступній), то перехід буде виконаний у межах тієї сторінки, де знаходиться другий байт команди. Для умовного переходу з однієї сторінки на іншу можна скористатися тандемом з команди умовного і безумовного переходів JMP.

Команда JMPR здійснює перехід за адресою, яка зберігається у пам'яті програм, на яку вказує вміст акумулятора. Таким чином, акумулятор містить адресу адреси переходу. Комірка з адресою переходу повинна бути на тій же сторінці пам'яті програм, що і команда переходу JMPR. Команда непрямого переходу забезпечує простий доступ до таблиці, що містить вектори переходів за програмою залежно від вмісту акумулятора, що дозволяє легко реалізувати механізм множинних розгалужень.

За командами умовних переходів можуть перевірятись не тільки ознаки, а й деякі сигнали на зовнішніх входах МК. Це дозволяє ефективно виконувати розгалуження у програмі, не використовуючи зайві команди введення та порівняння. Усі команди умовних переходів використовують пряму коротку адресацію, що накладає обмеження на розміщення програм у пам'яті.

Для організації циклу зручно використовувати команду DJNZ. Лічильник циклів організується в одному з регістрів поточного банку, у котрий завантажується число повторень. При виконанні команди DJNZ здійснюється декремент регістра та перевірка нового значення на рівність нулю. Якщо його вміст не нульовий, то відбувається перехід до початку циклу, інакше – вихід з циклу. Структура програми при цьому буде такою:

```
MOV RN, #N ;Ініціалізація лічильників циклу
```

```
LOOP: . . . ;Тіло циклу
```

```
. . .
```

```
DJNZ R1, LOOP; Декремент R1, та перехід, якщо не нуль
```

Команда DJNZ може здійснювати перехід лише у межах сторінки розміром 256 байтів.

Для виклику підпрограм існує команда CALL, що дозволяє звернутися у будь-яке місце поточного банку ПП. При виклику підпрограми у стеку запам'ятовується адреса повернення і частина РССП (чотири старші розряди). Глибина вкладень підпрограм обмежена ємністю стека (16 байт) і не повинна перевищувати восьми.

Для повернення з підпрограми використовують команду RET, що відновлює у лічильнику команд адресу повернення. Для виходу з підпрограми обслуговування переривання використовується команда RETR, що крім адреси повернення відновлює РССП і дозволяє переривання даного джерела.

Програмуючи мікроконтролер, необхідно уважно відслідковувати виклики підпрограм у альтернативному (відносно поточного) банку пам'яті програм. У цьому випадку перед викликом підпрограми необхідно вибрати відповідний банк пам'яті, а перед поверненням – відновити старе значення DBF. Якщо в підпрограмі немає команди відновлення DBF, то повернення буде виконане вірно, однак перша ж команда довгого переходу передасть керування в альтернативний банк пам'яті програм. Відновлення біта DBF можливо або у самій підпрограмі або відразу за командою CALL. У підпрограмі попереднє значення біта DBF визначається за старшим бітом адреси повернення.

Нехай, наприклад, потрібно викликати підпрограму у першому банку пам'яті програм з ім'ям SUBROUT, з програми у нульовому банку. Коректний виклик підпрограми реалізує послідовність команд:

```
SEL MB1          ;Встановлення ознаки першого банка  
CALL SUBROUT    ;Виклик підпрограми після повернення  
                ;з підпрограми DBF = 1  
SEL MB0          ;Відновлення DBF, щоб уникнути  
                ;помилкового переходу в банк 1
```

4.9. Команди керування режимом роботи лічильника-таймера

У групу входять команди завантаження, зчитування, керування лічильником-таймером та перериванням від лічильника-таймера. У таблиці 32

наведені формати команд цієї групи.

Таблиця 32. Група команд керування лічильником-таймером

Позначення	Код	T ^{*)}	B ^{*)}	C ^{*)}	Назва та дія команди
MOV A, T	10000100	1	1	1	Пересилання вмісту лічильника-таймера в акумулятор: A = T
MOV T, A	01100010	1	1	1	Пересилання акумулятора в лічильник-таймер: T = A
STRT T	01010101	1	1	1	Запуск таймера
STRT CNT	01000101	1	1	1	Запуск лічильника
STOP TCNT	01100101	1	1	1	Зупинка лічильника-таймера
TCNTI	00100101	1	1	1	Дозвіл переривання від лічильника-таймера
DIS TCNTI	00110101	1	1	1	Заборона переривання від лічильника-таймера

^{*)} У таблиці використані позначення: T – тип команди (за рис. 50), B – кількість байтів у коді команди, C – тривалість виконання команди у машинних циклах

До групи відносять команди обміну даними між таймером і акумулятором (MOV A,T і MOV T,A), за якими вміст лічильника можна прочитати під час відліку або під час зупинки лічильника-таймера або перезавантажити його. У OMEOM виконують спеціальні команди керування режимом роботи таймера. Таймер можна використати як лічильник тактів чи як лічильник подій від внутрішнього чи зовнішнього джерела сигналів. Система команд МК має засоби дозволу чи заборони переривання від таймера.

Спеціальна команда ENT0 на вихід T0 дозволяє передавати імпульси з частотою опорного сигналу, поділеною на три. Цей сигнал може від'єднаний лише скиданням. Сигнал на виході T0 використовується для загальної синхронізації зовнішніх пристроїв.

4.10. Команди перемикавання банків регістрів і та пам'яті програм

Перемикавання банків пам'яті програм, зміна старшого біта лічильника команд відбувається у момент виконання команди виклику підпрограми. Наявність команд перемикавання банків регістрів дозволяє при виклику підпрограм та виконанні підпрограм переривань ефективно використовувати другий банк регістрів як робочий, зберігаючи параметри обчислювального процесу не в стеку, а у вихідному банку регістрів. Програмуючи процедури переривань, можна перемикати чи не перемикати банки регістрів. У випадку,

коли банки регістрів перемикаються, повернення до вихідного банку регістрів буде виконано автоматично, якщо підпрограма переривань закінчується командою повернення з відновленням РССП (RETR).

Таблиця 33. Група команд керування контролером

Позначення	Код	Т ^{*)}	Б ^{*)}	Ц ^{*)}	Назва та дія команди
EN I	00000101	1	1	1	Дозвіл зовнішнього переривання
DIS I	00010101	1	1	1	Заборона зовнішнього переривання
SEL RB0	11000101	1	1	1	Вибір нульового банку регістрів: BS = 0
SEL RB1	11010101	1	1	1	Вибір першого банку регістрів: BS = 1
SEL MB0	11100101	1	1	1	Вибір нульового банку пам'яті програм: DBF = 0
SEL MB1	11110101	1	1	1	Вибір нульового банку пам'яті програм: DBF = 1
ENT0 CLC	01110101	1	1	1	Дозвіл видавання сигналу на вихід T0
NOP	00000000	1	1	1	Команда "немає операції"

^{*)} У таблиці використані позначення: Т – тип команди (за рис. 50), Б – кількість байтів у коді команди, Ц – тривалість виконання команди у машинних циклах

5. Програмування ОМЕОМ

5.1. Приклади використання команд передачі даних

Розглянемо використання окремих команд та їх послідовностей для виконання типових дій.

Приклад 1. Необхідно записати у внутрішню пам'ять даних у комірки з адресами 31h і 32h число 1C3Fh. Дії можна виконати такою послідовністю команд.

LOAD:

```
MOV R0, #31h ; завантаження в R0 покажчика
MOV @R0, #1Ch ; записування даних числа 1Ch
INC R0       ; збільшення покажчика адреси на 1
MOV @R0, #3Fh ; запис у пам'ять числа 3Fh
```

Приклад 2. Необхідно переслати поточний вміст таймера у R5 без втрати вмісту акумулятора:

XCHNG:

```
XCH A, R5    ; обмін R5 і акумулятора
MOV A, T     ; пересилання вмісту таймера в A
```

XCH A,R5 ;обмін R5 та акумулятора

Приклад 3. Передати вміст регістрів банку 0 у внутрішню пам'ять даних, починаючи з адреси 38h:

SEL RB1 ;вибір банку регістрів 1
MOV R0,#38h ;визначення початкової адреси
MOV R1,#0 ;визначення початкової адреси банку
MOV R2,#8 ;встановлення лічильника циклів

LP:

MOV A,@R1 ;пересилання байта з регістра у пам'ять
MOV @R0,A ;через акумулятор
INC R0 ;збільшення вказівників
INC R1
DJNZ R2,LP ;продовжити, якщо передані не всі дані

Приклад 4. Ввести байт із порту 1 і передати його в порт 2:

TRAN:

MOV A,#0FFh ;настроювання порту 1 на ввід
OUTL P1,A
IN A,P1 ;ввід байта з порту 1
OUTL P2,A ;вивід байта в порт 2

Приклад 5. Зчитати 8-бітні дані з пристрою, що під'єднаний до портів P7 та P6 та помістити їх в регістр A:

INPUT:

MOVD A,P7 ;пересилання чотирьох бітів з порту 7
;у молодшу тетраду акумулятора,
;старша тетрада стає рівною 0
SWAP A ;перенесення бітів у старші розряди
MOVD A,P6 ;зчитування молодших бітів
ORL A,R7 ;формування результату в регістрі A

Приклад 6. Написати підпрограму швидкого отримання значень функції значної складності від байтового аргумента за допомогою таблиці. Для

прикладу обчислимо значення функції $255 \cdot \sin\left(\frac{\pi(x+1)}{512}\right)$, де x знаходиться у регістрі R0. Для обчислення використовується таблиця функції для у всіх значень аргументу (всього 256 байтів). Необхідна таблиця функції займає одну сторінку пам'яті. Дану сторінку зручно розмістити на третій сторінці внутрішньої пам'яті програм:

```

sin_x:      ;обчислення A = 255*sin(pi*(x+1)/512)
MOV A,R0    ;пересилання аргументу в акумулятор
MOVP3 A,@A  ;завантаження в акумулятор значення
RET         ;повернення
ORG 300H    ;задання початкової адреси таблиці
;Таблиця значень функції. "DB" означає визначити байт
DB 02h, 03h, 05h, 06h, 08h, 09h, 0Bh, 0Dh ;
DB 0Eh, 10h, 11h, 13h, 14h, 16h, 17h, 19h ;
. . . . .
DB 0FEh, 0FEh, 0FEh, 0FEh, 0FEh, 0FFh, 0FFh, 0FFh ;
DB 0FFh, 0FFh, 0FFh, 0FFh, 0FFh, 0FFh, 0FFh, 0FFh ;

```

5.2. Приклад використання команд арифметичних операцій

Приклад 7. Необхідно додати до вмісту регістра R7 вміст комірки внутрішньої пам'яті даних з адресою 60:

```

MOV R0,#60   ;завантаження в R0 адреси
MOV A,R7     ;завантаження операнда в акумулятор
ADD A,@R0    ;додавання

```

Результат операції додавання фіксується в акумуляторі, встановлення ознаки перенесення C свідчитиме про переповнення.

Приклад 8. Додати двійково-десяткові (BCD) числа, розташовані в A та R7:

```

ADD A,R7     ;двійкове додавання
DA A         ;десяткова корекція A

```

Приклад 9. Провести інкрементування вмісту комірок пам'яті даних за

адресами 10÷18:

INCR:

MOV R0, #10 ; завантаження в R0 початкової адреси

MOV R3, # (18-10+1)

; завантаження в R3 числа комірок

LOOP:

INC @R0 ; інкремент комірок РПД

INC R0 ; збільшення вказівника адреси

DJNZ R3, LOOP ; декремент R3 та повторення, до R3=0

Приклад 10. Необхідно додати багатобайтові BCD-числа, розташовані в внутрішній пам'яті даних. Регістри R0 і R1 вказують початкові адреси доданків. Доданки розташовані, починаючи з молодших байтів. Формат доданків однаковий і задається в R2 кількістю байтів. Результат додавання необхідно розмістити на місце першого доданка:

CLR C ; скидання ознаки перенесення

LP:

MOV A, @R0 ; завантаження поточного байта

ADDC A, @R1 ; додавання з врахуванням перенесення

DA A ; корекція A

MOV @R0, A ; збереження поточного байта результату

INC R0 ; перехід до наступних байтів доданків

INC R1

DJNZ R2, LP ; декремент R2, та перехід на початок

; поки R2 не дорівнює 0

Операцію віднімання можна виконати заміною віднімання на додавання двома способами :

- ◇ можна число, що віднімається, перевести у додатковий код, а потім провести операцію додавання;
- ◇ зменшуване перевести у зворотний код, додати до другого операнда та проінвертувати суму.

Приклад 11. Виконати дію $A = R1 - A$. Провести віднімання першим методом.

```
CPL A          ;формування зворотного кода
ADD A,#01      ;формування додаткового кода
ADD A,R1       ;отримання результату віднімання
```

Приклад 12. Виконати дію $A = A - R1$. Провести віднімання другим методом.

```
CPL A          ;формування зворотного кода
ADD A,R1       ;формування додаткового кода
CPL A          ;повернення до прямого кода
CPL C          ;інверсія ознаки перенесення
```

Перший метод зручніший, коли у акумуляторі знаходиться число, що віднімається, другий – коли у акумуляторі зменшуване. Встановлення ознаки C після виконання додавання буде свідчити про від'ємний знак результату.

Приклад 13. Отримати різницю 2-байтнових чисел без знака. Операнди розташовуються у внутрішній пам'яті даних. Адреса зменшуваного зберігається в R1, а числа, що віднімається у R0. Результат віднімання розмістити на місці зменшуваного:

SUBSTR:

```
MOV A,@R0      ;завантаження молодшого байта Y
CPL A          ;отримання додаткового коду Y
INC A
ADD A,@R1      ;обчислення молодшого байта
MOV @R0,A      ;збереження молодшого байта різниці
INC R0         ;перехід до старших байтів чисел
INC R1
MOV A,@R0      ;завантаження старшого байта Y
CPL A          ;зворотний код Y
ADDC A,@R1     ;визначення старшого байта
MOV @R0,A      ;збереження старшого байта різниці
```

Приклад 14. Помножити однобайтові цілі числа без знака. У регістрі R1 розміщений множник, у регістрі R2 – множене. Двобайтовий результат множення буде розміщений в акумуляторі (старший байт) і в R1 (молодший байт). У регістрі R3, що виконує функції лічильника програмних циклів, завантажується число 8 (кількість бітів множника). Множення виконується молодшими бітами зі зсувом вправо часткового добутку. Послідовність дій при цьому методі множення така:

Вміст акумулятора і регістра R1 зсувають вправо на один біт так, що молодший біт множника, висунутий з регістра R1, попав у тригер ознаки C. Якщо $C = 1$, то множене додається до вмісту акумулятора, в іншому випадку ніяких операцій не здійснюється. Далі декрементується лічильник циклів R3, і якщо його вміст не дорівнює нулю, то усі дії повторюються. Перед виходом із підпрограми формується остаточний результат зсувом часткового результату на один біт вправо:

MPLY1B:

```
MOV R3, #8      ;завантаження лічильника циклів
CLR A           ;завантаження лічильника циклів
CLR C           ;очищення ознаки перенесення
```

SHIFT:

```
RRC A           ;зсув акумулятора вправо
XCH A, R1       ;обмін акумулятора і R1
RRC A           ;зсув множника вправо, молодший біт
                ;заноситься у C
XCH A, R1       ;обмін акумулятора і R1
JNC RESULT     ;якщо C = 1, то підсумовування
ADD A, R2       ;додавання множеного
```

RESULT:

```
DJNZ R3, SHIFT;декремент R3=0 та перевірка
RRC A           ;зсув акумулятора
XCH A, R1       ;обмін
```

```

RRC A          ;зсув вмісту R1
XCH A,R1       ;обмін

```

5.3. Приклади використання команд логічних операцій

Приклад 15. Маскування при введенні даних. Ввести в регістр R7 інформацію з ліній 0,1,3,4 і 7 порта P1:

```

IN A,P1        ;введення байта з порта P1
ANL A,#10011011B
                ;відкидання непотрібних бітів
MOV R7,A       ;пересилання результатів роботи у R7

```

Код #10011011B задає двійкове число 10011011₂ у якості аргументу команди логічного І.

Приклад 16. Ввести в акумулятор дані з порту P2 і виділити необхідні біти по масці, що знаходиться в R0:

```

IN A,P2        ;введення з порту P2
ANL A,R0       ;маскування розрядів

```

Приклад 17. Виконати зсув вліво через ознаку перенесення двобайтного слова, розташованого у R2 та A:

```

RLC A          ;зсув молодшого байта
XCH A,R2       ;обмін акумулятора та регістра
RLC A          ;зсув старшого байта
XCH A,R2       ;повернення молодшого байта

```

Приклад 18. Виконати циклічний зсув вправо двобайтного слова з регістрів R2 та A :

```

CLR C          ;скидання біта перенесення
JB0 Skip       ;якщо молодший біт A рівний нулю ознака
                ;C має теж дорівнювати нулю
CPL C          ;встановлення ознаки перенесення
Skip:
XCH A,R2       ;обмін байтами
RRC A          ;зсув старших бітів

```

```

XCH A,R2      ;обмін байтами
RRC A         ;зсув молодших бітів

```

Приклад 19. Виконати логічний зсув вправо двобайтового слова з регістрів R2 та A:

```

CLR C         ;скидання біта перенесення
XCH A,R2      ;обмін байтами
JB7 Skip      ;якщо молодший біт A рівний нулю ознака
               ;C має теж дорівнювати нулю
CPL C         ;встановлення біта перенесення
Skip:
RRC A         ;зсув старших бітів
XCH A,R2      ;обмін байтами
RRC A         ;зсув молодших бітів

```

Приклад 20. Помножити акумулятор на число 2 у степені X, де X – число (не більше 8), що зберігається в R2. Множення на 2 замінюють арифметичним зсувом вліво акумулятора та регістра старшого байта результату (R1):

```

MOV R1,#0     ;скидання R1
CLR C         ;скидання ознаки перенесення
LOOP:         ;арифметичний зсув вліво 16-бітного
; результату в регістровій парі R1, A
RLC A         ;зсув молодшого байта
XCH A,R1      ;обмін байтів
RLC A         ;зсув старшого байта
XCH A,R1      ;повернення байтів
DJNZ R2,LOOP;перевірка кінця цикла

```

Приклад 21. Видати вміст акумулятора в послідовному коді через нульову лінію порту 1, залишаючи без зміни інші біти. Передавання даних проводити, починаючи з молодшого біта:

```

MOV R1,#8     ;лічильник бітів
LOOP:

```

```

JB0 ONE      ;перехід, якщо біт A.0 дорівнює 1
ANL P1,#FEh  ;скидання P1.0
JMP NEXT

```

ONE:

```

ORL P1,#1    ;встановлення P1.0
JMP NEXT     ;додаткова команда для вирівнювання
              ;часу передавання нуля та одиниці

```

NEXT:

```

RR A         ;підготовка до передачі чергового біта)
DJNZ R1,LOOP

```

Приклад 22. Підготувати біти 0-3 порту P1 для введення даних:

```

ORL P1, #0FH;встановлення бітів P1.0...P1.3

```

Приклад 23. Очистити біти 4-7 порту 2:

```

ANL P2,#0FH  ;скидання бітів P2.4...P2.7

```

Приклад 24. Видати у лінію 0 порта 4 значення нульового біта акумулятора:

```

ANL A,#1     ;виділення біта A.0
ORLD P4,A    ;встановлення P4.0, якщо A.0=1
ORL A,#0EH   ;встановлення в 1 бітів 1÷3 акумулятора
ANLD P4,A    ;скидання P4.0, якщо A.0=0

```

Приклад 25. Визначити парність числа одиниць в акумуляторі:

```

CLR F0       ;скидання F0
MOV R7,#8    ;число повторів
LOOP:
RRC A        ;пересилання біта A.0 у біт C
JNC NEXT     ;пропустити, якщо біт дорівнює 0
CPL F0       ;підрахунок паритету

```

NEXT:

```

DJNZ R7,LOOP;повторити для всіх бітів байта

```

Після виконання програми акумулятор збереже своє значення, ознака F0

буде встановлена в 1, якщо число одиниць в акумуляторі було непарним. Ознака F0 входить до складу PSW і в даному прикладі використовується для виконання функції ознаки паритету.

5.4. Приклади використання команд передавання керування і команд керування режимом МК48

Приклад 26. Передати керування на мітку LL, якщо перемикач банків регістрів (біт PSW.4) встановлений:

JBSET:

```
MOV A, PSW      ;передавання PSW в акумулятор
JB4 LL          ;перехід, якщо A.4=1
```

Приклад 27. Передати керування на мітку LABEL, якщо лічильник подій досягне значення 64:

TESTC:

```
MOV A, T        ;пересилання вмісту лічильника в A
JB6 LABEL       ;Перехід за міткою, якщо A.6=1
```

Приклад 28. Здійснити перехід з нульового банку ПП до підпрограми з ім'ям ROUT, розташованої в першому банку ПП:

```
SEL MB1         ;встановлення ознаки DBF
CALL ROUNT      ;перехід до програми ROUNT
SEL MB0         ;відновлення ознаки DBF
```

Приклад 29. Множинне розгалуження програми. Припустимо, що результатом роботи деякої програми є число X (у межах від 0 до 15). Необхідно організувати переходи на 16 різних програм з іменами ROUT0 ÷ ROUTF залежно від обчисленого значення X:

```
ANL A, #0FH     ;скидання старшої тетради A
JMPP @A         ;перехід за таблицею векторів переходів
;Таблиця векторів переходів
DB ROUT0        ;початкова адреса програми ROUNT0
DB ROUT1        ;початкова адреса програми ROUNT1
```

```

    DB ROUT2      ;початкова адреса програми ROUNT2
    . . .
    DB ROUTF      ;початкова адреса програми ROUNTF

```

Слід наголосити, що команда JMPP, таблиця переходів та програми ROUT0 ÷ ROUTF повинні знаходитися на одній сторінці пам'яті програм.

Приклад 30. Уникнути обмежень на розміщення адрес можна змінюючи адресу повертання записану у стек. Наведений фрагмент здійснює перехід на адресу задану у регістрах R1 та R0. Адреса не має перемикати банк пам'яті програм.

```

    CALL JMP_REL ;збереження у стеку адреси JMP_REL
JMP_REL:
    MOV A,PSW      ;зчитування слова стана
    DEC A          ;узгодження вказівника й вершини стека
    ANL A,#07      ;відкидання непотрібних бітів
    RL A           ;перехід до зміщення вершини у байтах
    ADD A,08        ;визначення адреси вершини стека
    XCH A,R0        ;сформована адреса зберігається у R0
    MOV @R0,A       ;зміна молодшого байта адреси
                   ;повернення
    INC R0          ;перехід до старшого байта
    MOV A,R1        ;підготовка старшого байта
    MOV @R0,A       ;його збереження
    RET            ;команда повернення здійснить перехід
                   ;за адресою, що була у регістрах R1 R0

```

Слід зауважити, що фрагмент використовує додатковий елемент стека, але звичайно такі операції виконуються на основному рівні програмі, де стек далекий від заповнення.

Приклад 31. Організувати чекання появи нульового рівня на вході T1:

```

WAIT:  JT1 WAIT ;перехід на WAIT, якщо на вході T1 0

```

Аналогічним чином можна перевірити стан ліній T0 та INT.

Приклад 32. Передавання керування одній з восьми програм ROUT0÷ROUT7 за появою нульового рівня на відповідному вході порту BUS. Найвищим пріоритетом володіє вхід BUS.0:

```
INS A, BUS ;встановлення режиму вводу через BUS
LOOP:
INS A,P1   ;ввід даних з через BUS
CPL A      ;Інверсія акумулятора
JZ LOOP    ;Чекання появи першого нуля
JB0 ROUT0  ;перехід до програми ROUNT0, якщо BUS.0=0
JB1 ROUT1  ;перехід до програми ROUNT1, якщо BUS.1=0
JB2 ROUT2  ;перехід до програми ROUNT2, якщо BUS.2=0
JB3 ROUT3  ;перехід до програми ROUNT3, якщо BUS.3=0
JB4 ROUT4  ;перехід до програми ROUNT4, якщо BUS.4=0
JB5 ROUT5  ;перехід до програми ROUNT5, якщо BUS.5=0
JB6 ROUT6  ;перехід до програми ROUNT6, якщо BUS.6=0
JB7 ROUT7  ;перехід до програми ROUNT7, якщо BUS.7=0
```

Розгалуження здійснюється групою з восьми команд JBb. Пріоритети входів BUS визначаються черговістю перевірки.

Приклад 31. При надходження на вхід T0 послідовності з восьми імпульсів встановити вихід P2.7:

```
MOV R7,#8 ;завантаження в R7 числа імпульсів
ONE:
JT0 ONE   ;чекання сигналу 0 на вході T0
ZERO:
JT0 SKIP  ;чекання сигналу 1 на вході T0
JMP ZERO
SKIP:
DJNZ R7,ONE;повторювати, поки не надійде 8 імпульс
ORL P2,#80H;встановлення одиниці на вході 7 порту 2
```

Тривалість нуля й одиниці на вході пристрою повинна бути не менш

чотирьох машинних циклів, тобто 10 мкс.

Приклад 33. Заборонити переривання від таймера, але дозволити переривання після восьми сигналів переповнення таймера. При переході до процедури обробки переривання зупинити таймер. Сигнали переповнення підраховувати в регістрі R5:

```
STARS:  DIS TCNT1;заборона переривань від таймера
        CLR A          ;скидання акумулятора
        MOV T,A        ;скидання таймера
        MOV R5,A       ;скидання регістра R5
        STRT T         ;запуск таймера
M:JTF COUNT      ;якщо TF=1, то перехід до COUNT та
                ;скидання TF
        JMP M1         ;цикл
COUNT:
        INC R5         ;інкремент регістра R5
        MOV A,R5       ;пересилання вмісту R5 в акумулятор
        JB3 INT        ;перехід до обслуговування
                ;переривання INT, якщо біт A.3 = 1
        JMP M1         ;перехід, якщо біт A.3 не дорівнює 1
        . . .
        . . .
INT:     STOP TCNT     ;зупинка таймера
        JMP 07H        ;перехід до комірки 7
```

Контрольні запитання

1. Однокристалні мікроЕОМ сімейства МК48. Основні характеристики.
2. Архітектура ОМЕОМ К1816ВЕ48.
3. Арифметико-логічний пристрій.
4. Організація резидентної пам'яті даних та пам'яті програм.
5. Система переривань.
6. Лічильник-таймер.

7. Порти вводу-виводу.
8. Функціонування МК48.Режими роботи. Передача сигналів через порти.
9. Адресний простір пам'яті.
10. Під'єднання до МК48 зовнішньої пам'яті даних та програм.

Архітектура однокристальних мікро-ЕОМ сімейства МК-51

1. Однокристальні мікро-ЕОМ сімейства МК-51

ОМЕОМ сімейства МК51 (MCS51) відносять до пристроїв з 8-розрядною організацією. Розроблені по n-МОН технології, ОМЕОМ зберегли свою архітектуру при переході на технологію КМОН, що дозволило на порядок знизити енергоспоживання, зберігши повну наступність апаратних і програмних засобів. В даний час серійно випускають лише ОМЕОМ, виконані за КМОН технологією.

Базовий кристал сімейства i80C51 має такі характеристики:

- ◇ об'єм резидентної пам'яті програм – 4 кбайт;
- ◇ тип резидентної пам'яті програм – репрограмоване ПЗП РПЗП;
- ◇ об'єм резидентної пам'яті даних – 128 байт;
- ◇ мінімальна частота тактових сигналів 1,2 МГц;
- ◇ максимальна частота тактових сигналів 12 МГц;
- ◇ напруга живлення, $+5\text{ В} \pm 10\%$;
- ◇ струм споживання, 8 мА;
- ◇ обсяг зовнішньої адресованої пам'яті програм, 64 кбайт;
- ◇ обсяг зовнішньої адресованої пам'яті даних, 64 кбайт;
- ◇ система команд ОМЕОМ i80C51 містить 111 базових команд;
- ◇ багаторівнева система переривань підтримує переривання від п'яти джерел. Мікросхема має два зовнішніх входи запитів переривань.

Дуже корисна особливість i80C51 – обробка бітових даних, що дозволяє використовувати бінарну логіку, котра оперує бітами внутрішнього ОЗП та регістрів. Ця особливість широко використовується у промисловій автоматичі. Ще одна корисна особливість – чотири незалежні набори регістрів. Які дуже значно зменшують затримки при обслуговуванні переривань.

До складу ОМЕОМ i80C51 входять такі додаткові пристрої:

- ◇ 4 восьмирозрядні паралельні порти вводу-виводу;
- ◇ два 16-розрядні лічильники-таймери;
- ◇ послідовний порт;
- ◇ тактовий генератор;
- ◇ блок регістрів спеціальних функцій;
- ◇ система захисту програм від несанкціонованого доступу.

У позначенні ОМЕОМ МК-51 цифри мають таке призначення:

- ◇ перша цифра показує розрядність АЛП (8 біт);
- ◇ друга цифра задає тип внутрішньої пам'яті програм: 0 – ПЗП масочного типу, 3 – ПЗП з зниженою напругою живлення (+3В); 7 – репрограмоване з ультрафіолетовим стиранням, 9 – репрограмоване ПЗП з електричним стиранням,
- ◇ літера указує технологію виготовлення: С – пристрій, виконаний за КМОН технологією, відсутність літери – указує n-МОН технологію;
- ◇ останні дві цифри задають код сімейства: 51 – МК51 (MCS51), 31 – варіант без внутрішнього ПЗП програм.

Якщо перед зазначеним позначенням є літери, то вони означають конструктивне виконання:

- ◇ D – керамічний корпус DIP 40 виводів;
- ◇ P – пластиковий корпус DIP 40 виводів;
- ◇ N – корпус PLCC, 44 виводи.

Керамічний корпус DIP має вбудоване вікно, закрите кварцовим склом, для стирання ультрафіолетовим випромінюванням записаної в пам'ять програм. Пластиковий корпус DIP вікна не має, тому ОМЕОМ P87C51 вважається однократно програмованим виробом, призначеним для дрібносерійного виробництва. Випускають також мікросхеми з підвищеною швидкодією – із граничним значенням тактової частоти 16, 20 і 24 МГц. Крім базового виробу, до складу сімейства МК51 входять такі модифікації:

- ◇ 87C52, 87C54 і 87C58, котрі мають внутрішнє ОЗП даних ємністю 256 байтів, пам'ять програм ємністю відповідно 8, 16 та 32 кбайти, а

також додатковий лічильник-таймер;

◇ 87C51FA, 87C51FB, 87C51FC, аналогічні за характеристиками, 87C52, 87C54, 87C58, та мають у своєму складі кілька програмованих лічильників-таймерів. Таймери забезпечують функції типу ШІМ, прискореного послідовного виведення та інш.;

◇ 87C51GB, що має ОЗП даних ємністю 256 байтів та ПЗП ємністю 8 кбайтів, 8 аналогових входів та 16 рівноцінних портів вводу-виводу.

З ОМЕОМ сімейства МК51, які випускають інші фірми, на особливу увагу заслуговують вироби фірми Atmel:

◇ AT89C51, яке містить ПЗП з електричним стиранням. Мікросхема має підвищену навантажувальну здатність портів, та забезпечує пряму роботу з світлодіодами:

◇ AT89C52, аналогічні 87C52, але з ПЗП з електричним стиранням;

◇ AT89C1051 і AT89C2051, що мають ПЗП з можливістю програмування ємністю, 1 чи 2 кбайти, вбудований компаратор аналогових сигналів, один чи два лічильники-таймери. Особливою відмінністю мікросхем є скорочене число виводів. Мікросхеми мають 15 програмованих ліній вводу-виводу з підвищеною навантажувальною здатністю.

Усі вироби фірми Atmel працюють при живленні від джерела напруги 2,7...6 вольт і при зміні частоти тактового генератора, що задається в діапазоні від 0 до 24 МГц.

Серед інших виробів сумісних з МК-51 варто відзначити сімейство C8051F12x фірми Cygnal. Сімейство C8051F12x – це i80C51 сумісні мікроконтролери з дуже високою продуктивністю, що досягає 10^8 операцій у секунду, великим об'ємом пам'яті та вбудованою аналого-цифровою периферією. На відміну від базових мікросхем, ядро C8051F12x виконує одну операцію за один такт та працює при тактовій частоті 100 МГц. Тобто еквівалентна тактова частота C8051F12x дорівнює 1.2 ГГц.

2. Архітектура OMEOM i80C51

Архітектура розглянутого сімейства OMEOM МК51 не може бути віднесена ні нейманівського, ні до гарвардського типів у їх класичній інтерпретації.

Пам'ять програм і пам'ять даних в OMEOM сімейства МК-51 не тільки фізично і логічно розділені, вони також мають різні тип і систему адресації. Водночас, для звертання до інформації, що зберігається у пам'яті даних та програм, використовується єдина восьмирозрядна шина, що значною мірою сповільнює процес обміну даними.

Використання єдиної шини для звертання до пам'яті даних і пам'яті програм є для OMEOM i80C51 компромісною мірою. Воно дозволяє ціною зниження продуктивності спростити апаратні засоби при роботі з зовнішньою пам'яттю і тим самим знизити вартість МП системи.

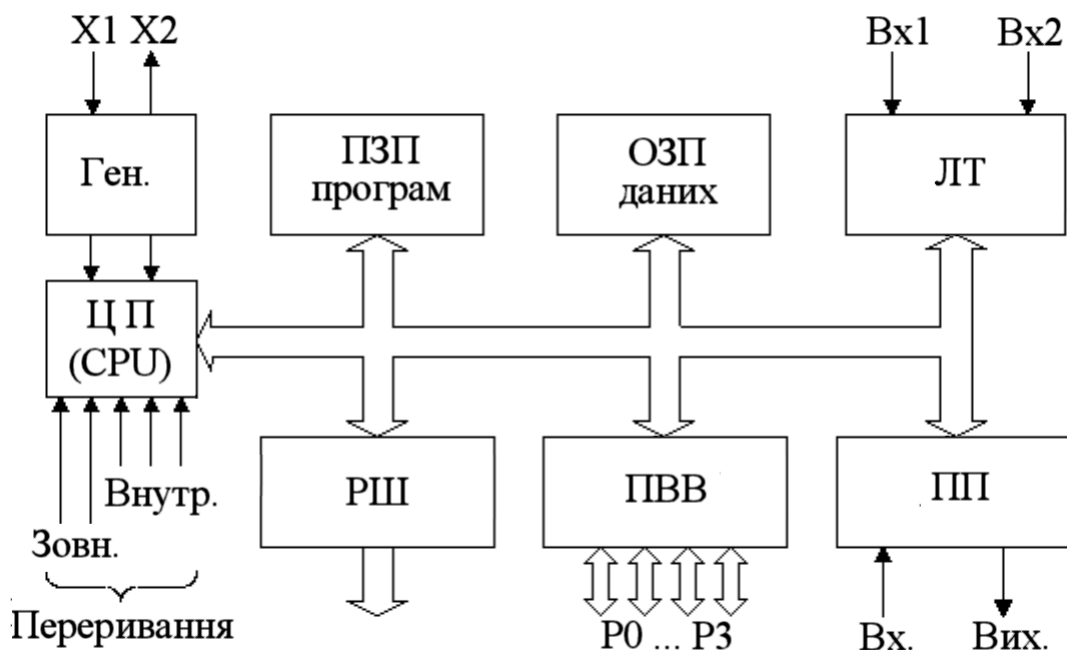


Рис. 52. Загальна структура OMEOM

Загальна структура OMEOM i80C51 зображена на рис. 52. До складу OMEOM входять такі функціональні вузли:

- ◇ ЦП – центральний процесор (англ. CPU – Central Processor Unit);
- ◇ ПЗП програм – постійний запам'ятовуючий пристрій призначений для збереження програми користувача;

- ◇ ОЗП даних – оперативний запам'ятовуючий пристрій, який використовується для збереження даних;
- ◇ Ген. – генератор тактових сигналів;
- ◇ ПВВ – порти вводу-виводу;
- ◇ ПП – послідовний порт з програмованим режимом та швидкістю роботи;
- ◇ ЛТ – лічильники-таймери, два 16-розрядних лічильники з можливістю відліку часу;
- ◇ РШ – розширювач шини для роботи з зовнішньою пам'яттю ємністю до 64 кбайт.

Усі вузли пов'язані між собою загальною восьмирозрядною шиною. ЦП виконує програму, записану в ПЗП програм. ЦП забезпечує виконання таких груп операцій:

- ◇ арифметичні: додавання, додавання з урахуванням перенесення, віднімання з урахуванням позичання, беззнакове множення і ділення, інкремент і декремент, десяткова корекція;
- ◇ логічні: І, АБО, виключне АБО, інверсія;
- ◇ зсуви;
- ◇ пересилання даних;
- ◇ робота з окремими бітами;
- ◇ переходи та виклики підпрограм.

Сам ЦП розділений на керуючий пристрій КП й операційний пристрій ОП. Проміжні результати обчислень зберігаються в ОЗП даних (RAM) ємністю 128 байтів. Крім того, в адресному просторі ПД розміщені всі регістри спеціальних функцій, які задають стан портів, таймери, регістр акумулятор (А), регістр В, регістр вказівника стека SP, з блоку РШ в адресний простір ПД входить вказівник даних DPTR (англ. Data PoinTeR) з асоційованими шинами обміну і лічильником команд PC (англ. Program Counter), призначення якого полягає у формуванні адреси команди. Крім того, у ОЗП розміщується стек, у якому можуть зберігатись адреси повернення з підпрограм та локальні змінні.

Швидкість роботи ЦП задається генератором Ген., що виробляє необхідні для роботи часові послідовності. Тактова частота Ген. задається або кварцовим резонатором, що вмикається між виводами X1 і X2, або зовнішнім генератором, що під'єднується до входу X1. З метою забезпечення послідовного доступу до ресурсів процесора при використанні однієї шини Ген. формує машинний цикл процесора з дванадцяти тактів резонатора.

Машинний цикл містить 6 станів керуючого автомата S1...S6, кожен стан розділений на дві фази P1, P2, що відповідає різним діям процесора.

Введення чи ведення даних у процесі обробки інформації, може бути здійснений або в паралельній байтовій (звід восьми розрядів однією командою), або в послідовній (по одному біту) формах. Паралельний обмін інформації можливий через один із чотирьох підтримуваних ОМЕОМ ПВВ. Послідовний обмін інформацією може бути організований через будь-який з розрядів ПВВ, однак для полегшення процесу послідовного обміну й економії обчислювальних ресурсів, необхідних для його реалізації, ОМЕОМ містить убудований програмований послідовний порт, що дозволяє практично без витрат обчислювальних ресурсів організувати послідовний обмін за кількома видами протоколів.

Крім розглянутих вузлів, до складу ОМЕОМ входять два шістнадцятирозрядні таймери-лічильники, що можуть функціонувати або в режимі таймера, або в режимі лічильника зовнішніх подій. Режим таймера використовують, коли необхідно організувати циклічні процеси з жорстко фіксованим і незалежним від часу виконання програми періодом циклу, наприклад, при опрацюванні сигналів, коли необхідно забезпечити необхідний інтервал дискретизації. Режим лічильника зовнішніх подій використовується, наприклад, при підрахунку кількості імпульсів, вимірюванні частоти і т.п.

Розширювач шини РШ використовується для роботи з зовнішнім запам'ятовуючими пристроями пам'яті програм чи даних. Як правило, зовнішні запам'ятовуючі пристрої використовуються, для розміщення програми чи даних у випадку нестачі внутрішніх ресурсів. Режим роботи з зовнішньою пам'яттю

не є типовим для OMEOM.

Режим звертання до зовнішньої пам'яті, використовується не тільки за прямим призначенням. Зовнішня шина універсальна, тому до неї можна під'єднати будь-який спеціалізований пристрій, ВІС комплектів КР580 і КР1810 або спеціалізовану мікро-ЕОМ і керувати їх роботою безпосередньо з і80С51.

Докладніше архітектура OMEOM і80С51 зображена на рис. 53. Вона буде використовуватися далі для пояснення різних аспектів роботи OMEOM.

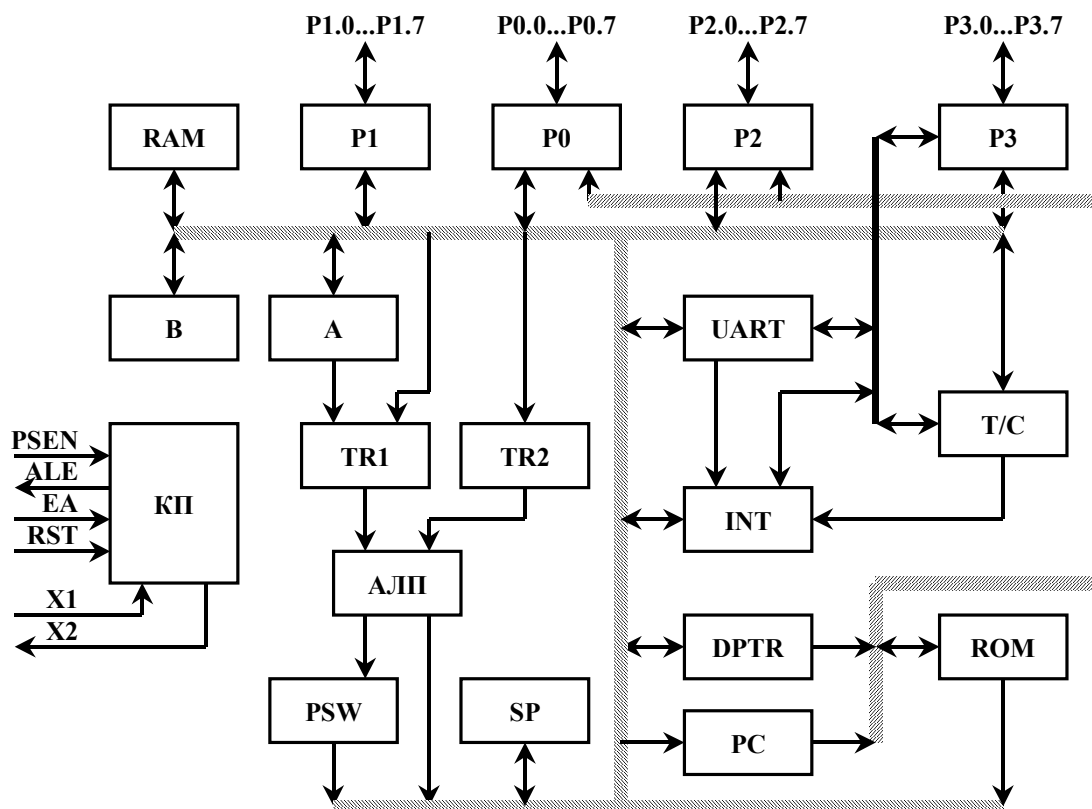


Рис. 53. Структурна схема і87С51

На рис. 53 вузол центрального процесора розділений на керуючий пристрій КП і чотири блоки, що є складають операційний пристрій: арифметико-логічний пристрій (АЛП), регістри тимчасового збереження операндів TR1 і TR2 та регістр ознак PSW (від англ. Program Status Word).

Крім того, з RAM виділені три регістри спеціальних функцій: акумулятор (A), регістр B і регістр показчика стека SP. Блок розширювача шини (ПШ) представлений показчиком даних DPTR (від англ. Data Pointer) з асоційованими шинами обміну. Лічильник команд PC (від англ. Program Counter) призначений для формуванні адреси наступної команди.

При використанні зовнішньої пам'яті адреса комірки передається через лінії портів P0 (молодший байт) та P2 (старший байт). Адреса поступає на порти по спеціальній шині з DPTR або PC. Обмін даними також здійснюється через порт P0. Після завершення обміну лінії портів P2 та P0 повертаються у попередній стан.

3. Регістр ознак

У процесі виконання операцій формуються ознаки: перенесення C, додаткового перенесення (перенесення між тетрадами) AC, нульового значення вмісту акумулятора Z, переповнення OV, парності P.

Для збереження ознак використовується регістр PSW. Його структура зображена на рис. 54. На відміну від інших ознак, OMEOM сімейства МК51 не зберігають ознаки Z, тому він інтерпретується не як ознака нульового результату операції, а як ознака нульового значення вмісту акумулятора.

D7	D6	D5	D4	D3	D2	D1	D0
C	AC	F0	RS1	RS0	OV	–	P

Рис. 54. Формат регістра PSW

На ознаку C впливає виконання команд: ADD, ADDC, SUBB, MUL, DIV, RLC, RRC, CJNE та операції над бітами. На ознаку OV впливають виконання команд MUL і DIV. Ознака парності характеризує поточний стан акумулятора, доповнюючи до парного кількість одиниць у його вмісті. Формально, займаючи розряд D0 у регістрі ознак PSW, він може бути переданий у стек, однак не підлягає ні встановленню в одиницю, ні скиданню в 0 без зміни вмісту акумулятора.

Наявність у PSW ознаки P дає можливість здійснити за його значенням умовні переходи за допомогою команд: JB P, addr, JNB P, addr.

Ознака AC у сукупності з командою DA A призначені для виконання програм, що реалізують дії у десятковій системі числення.

Розряд D5 у регістрі ознак є ознакою користувача F0 і може використовуватися на розсуд програміста. Біт разом з іншими записується у

стек разом з іншими елементами PSW.

4. Умовне позначення мікросхеми

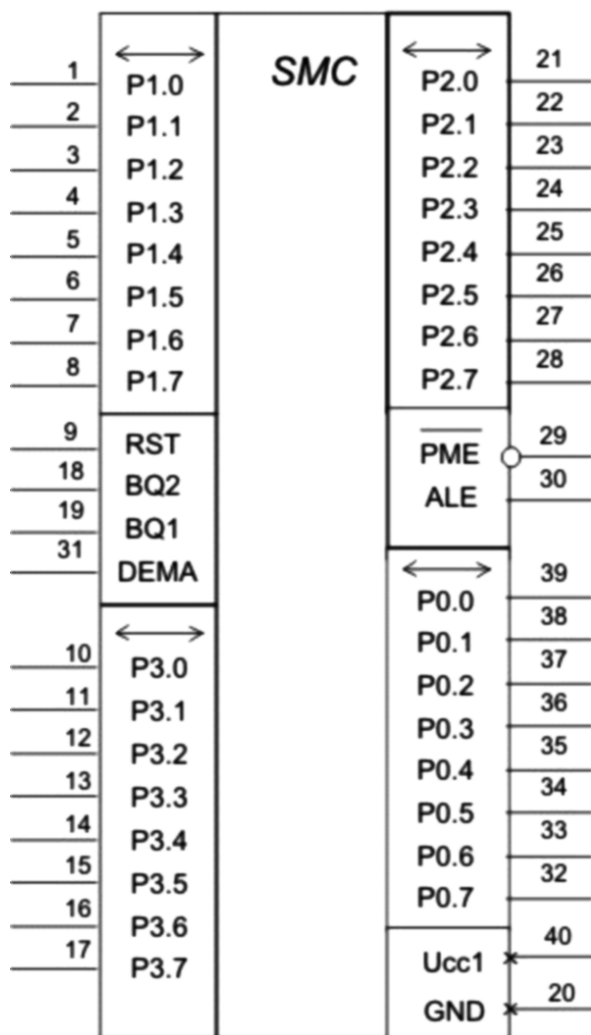


Рис. 55. Умовне позначення i80C51

Позначення OMEOM на електричній схемі зображено на рис. 55, а призначення виводів наведено в таблиці 34.

Слід зауважити, що на відміну від МК-48 в мікро-ЕОМ цього сімейства більшість виводів має подвійне функціональне призначення. Наприклад, при використанні зовнішнього ОЗП його адресація здійснюється через виводи портів P0 (молодший байт) і P2 (старший байт), а обмін інформацією (ввід кодів команд, ввід-вивід даних) – через виводи порта P0 при цьому керування обміном здійснюється через лінії P3.6 та P3.7. Тому, якщо не

застосовувати додаткові апаратні засоби, ці порти за наявності зовнішнього ОЗП не можна використати з іншою метою.

Лічильники-таймери, послідовний порт і система переривань не мають своїх виводів з корпусу OMEOM, а використовують виводи порту P3, і, таким чином, використання лічильників-таймерів у режимі лічильників зовнішніх подій, ліній послідовного порту і зовнішніх переривань знижує розрядність порту P3.

Специфічну для даної лінії функцію, яку виконує порт, називають альтернативною функцією лінії порту. А функцію простого вводу-виводу

називають основною функцією порту.

Зауважимо, що лінії портів P1÷P3 є квазідвонаправленими – фактично вони працюють як виводи з відкритим колектором, у яких внутрішні підтягуючі до високого рівня резистори знаходяться в середині мікро-ЕОМ. При встановленні відповідного біта порту в стан логічної 1 вивід видає невеликий струм, що обмежується підтягуючим резистором. При лог. 0 вивід приймає досить значний струм. Така асиметрія полегшує узгодження виводів з входами і виходами ТТЛ мікросхем.

Лінії порту P0 аналогічні лініям портів P1÷P3, за винятком того, що внутрішні підтягуючі резистори в них взагалі відсутні – лінії можуть лише приймати струм у стані логічного нуля. Проте при використанні ліній як виходів навантажених на входи мікросхем ТТЛ внутрішньої підтяжки входу достатньо для нормальної роботи виводів.

При зчитуванні стана лінії можливі два типи команд: команди, що зчитують стан лінії (команди вводу з порту), або команди, що зчитують стан внутрішнього регістра. При зчитуванні стана лінії необхідно, щоб на відповідному виході ОМЕОМ був заздалегідь виставлений лог. 1. У протилежному випадку лінія буде "посаджена" в стан лог. 0 і цей нуль і буде зчитаний.

Таблиця 34. Призначення виводів мікросхеми

Позначення	Номер виводу	Призначення
1	2	3
RST	9	Сигнал скидання мікросхеми або вхід напруги програмування внутрішнього ПЗП. Логічний 0 дозволяє нормальну роботу мікросхеми, логічна 1 протягом двох машинних циклів переводить мікросхему у режим скидання, рівень напруги 12 В на виводі призводить до програмування мікросхеми
BQ2 (X2)	18	Вивід для під'єднання зовнішнього резонатора
BQ1(X1)	19	Вивід для під'єднання зовнішнього резонатора
PME	29	Сигнал читання зовнішньої пам'яті програм. Логічний 0 на лінії вказує, що відбувається читання зовнішньої пам'яті програм

1	2	3
ALE	30	При нормальній роботі це вихід стробуючого сигналу адреси. Перехід сигналу зі стану лог. 1 в стан лог. 0 вказує на наявність адреси на лініях порту 0. У режимі програмування лінія є входом, на який надходить імпульс програмування
DEMA	31	Вхід заборони роботи внутрішньої пам'яті програм. Логічна 1 на вході дозволяє звертання до внутрішньої пам'яті програм. При нулю на вході при звертанні по адресах, що відповідають внутрішній пам'яті програм, звертання відбувається до зовнішньої пам'яті програм
P0.0-P0.7	39-32	Порт вводу-виводу 0, лінії порту також використовуються при звертанні до зовнішньої пам'яті програм та даних, а також до пристроїв вводу-виводу для виведення молодшого байта адреси і та обміну даними
P1.0-P1.7	1-8	Порт вводу-виводу 1. Виводи паралельного порту P1
P2.0/(A8)- P2.7/(A16)	21-28	Порт вводу-виводу 2, лінії порту також використовуються при адресації зовнішньої пам'яті та пристроїв вводу-виводу для виведення старшого байта адреси
P3.0/RxD	10	Лінія 0 порту вводу-виводу P3 може працювати як вхід даних у послідовному коді
P3.1/TxD	11	Лінія 1 порту вводу-виводу P3 може працювати як вихід даних у послідовному коді
P3.2/ $\overline{\text{INT0}}$	12	Лінія 2 порту вводу-виводу P3 може працювати як вхід запиту переривання по переходу сигналу в стан лог. 0
P3.3/ $\overline{\text{INT0}}$	13	Лінія 3 порту вводу-виводу P3 може працювати як вхід запиту переривання по переходу сигналу в стан лог. 0
P3.4/T0	14	Лінія 4 порту вводу-виводу P3 може працювати як вхід відліку внутрішнього лічильника 0
P3.5/T1	15	Лінія 5 порту вводу-виводу P3 може працювати як вхід відліку внутрішнього лічильника 1
P3.6/ $\overline{\text{WR}}$	16	Лінія 6 порту вводу-виводу P3 може працювати як сигнал записування у зовнішній ЗП даних, або порт вводу-виводу
P3.7/ $\overline{\text{RD}}$	17	Лінія 7 порту вводу-виводу 3 може працювати як сигнал читання зовнішнього ЗП даних, або порту вводу-виводу
GND	20	Напруга 0 В
Vcc1	40	Напруга живлення 5В

5. Мінімальна схема включення

Мінімальна схема ввімкнення OMEOM i80C51 зображена на рис. 56. До виводів X1 і X2 OMEOM під'єднана резонансна схема, що містить кварцовий резонатор. До входу скидання RST під'єднана схема автоматичного рестарту OMEOM при ввімкненні живлення. Коло з ємності та опору задає час дії сигналу скидання. Діод VD1 забезпечує скидання мікросхеми при короткочасному пропаданні живлення. Встановлення лінії DEMA у стан логічної 1 дозволяє зчитування кодів програми із внутрішнього ПЗП програм.

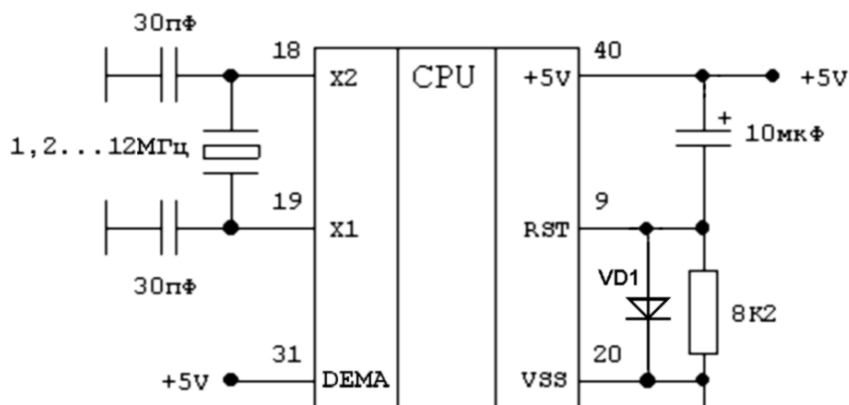


Рис. 56. Мінімальна схема ввімкнення

6. Загальна структура пам'яті OMEOM i80C51

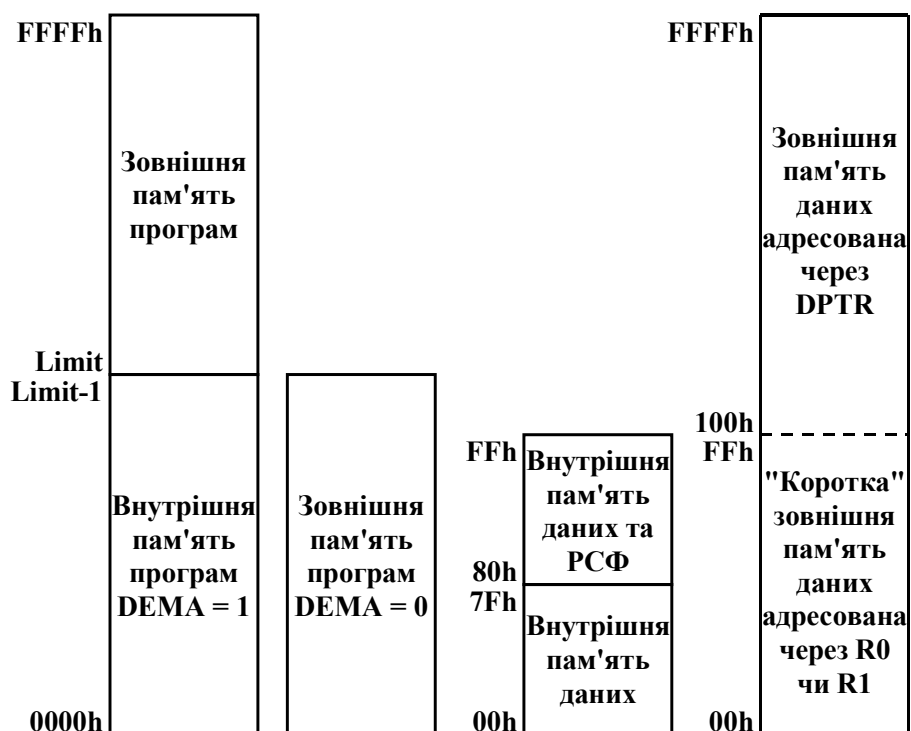


Рис. 57. Загальна структура пам'яті

В архітектурі i80C51 виділені внутрішня та зовнішня пам'яті даних, а також пам'ять програм. Пам'ять програм має максимальний розмір 64 кбайти. Вона може бути повністю зовнішньою, якщо сигнал DEMA рівний нулю, або якщо мікроконтролер сімейства немає внутрішньої пам'яті програм. Якщо ж пам'ять є, а сигнал DEMA рівний одиниці, звертання за адресами менше граничної (Limit) призводить до зчитування кодів із внутрішньої пам'яті програм. Звертання за адресами, що перевищують граничну призводять до зчитування зовнішньої пам'яті програм. Гранична адреса Limit залежить від об'єму пам'яті програм мікросхеми. У i80C51 вона дорівнює 1000h, а внутрішня пам'ять програм має об'єм 4 кбайти. Мікроконтролер i87C51 та більшість інших має 3 біти захисту коду, що захищають внутрішню пам'ять програм від доступу. При встановлених бітах захисту забороняється читання внутрішньої пам'яті ззовні, а також її допрограмування.

Пам'ять даних розділяється на внутрішню та зовнішню. Кожна з них має свій адресний простір. У архітектурі i80C51 на адресний простір внутрішнього ОЗП даних відображені усі програмно доступні блоки, які носять назву регістрів спеціальних функцій (РСФ).

Зовнішня пам'ять даних може використовуватись у одній з типових двох конфігурацій. У конфігурації з "короткою" адресацією для завдання номера комірки використовується 8-розрядний регістр, а вся пам'ять має об'єм 256 байтів. Зазвичай така конфігурація використовується для звертання до портів вводу-виводу. У конфігурації з повною адресацією для завдання номера комірки використовується 16-розрядний регістр DPTR, пам'ять має об'єм 64 кбайти. Зазвичай така конфігурація використовується для звертання до ОЗП.

У випадку об'єднання сигналів PМЕ та RD за допомогою логічного І можна отримати мікропроцесорну систему з об'єднаною пам'яттю для даних та програм майже класичної неймановської архітектури. Такі системи мають можливість модифікації зовнішньої пам'яті програм по програмі виконуваний ОМЕОМ. Проте, внутрішня пам'ять програм лишається незмінною, а звертання до зовнішньої пам'яті програм з адресами менше Limit неможливе. Для

вмикання можливості модифікації усієї пам'яті програм необхідно відключити внутрішню пам'ять програм.

7. Структура внутрішньої пам'яті даних

Структура внутрішньої пам'яті даних та регістрів відображених на неї зображена на рис. 58. Внутрішня пам'ять даних OMEOM i80C51 містить два функціонально розділених блоки – оперативний запам'ятовуючий пристрій (ОЗП), ємністю 128÷256 байтів, призначений для збереження даних, і блок регістрів спеціальних функцій РСФ, що містить набір регістрів, які встановлюють режими роботи деяких вузлів OMEOM. Призначення РСФ буде розглянуто у подальшому.

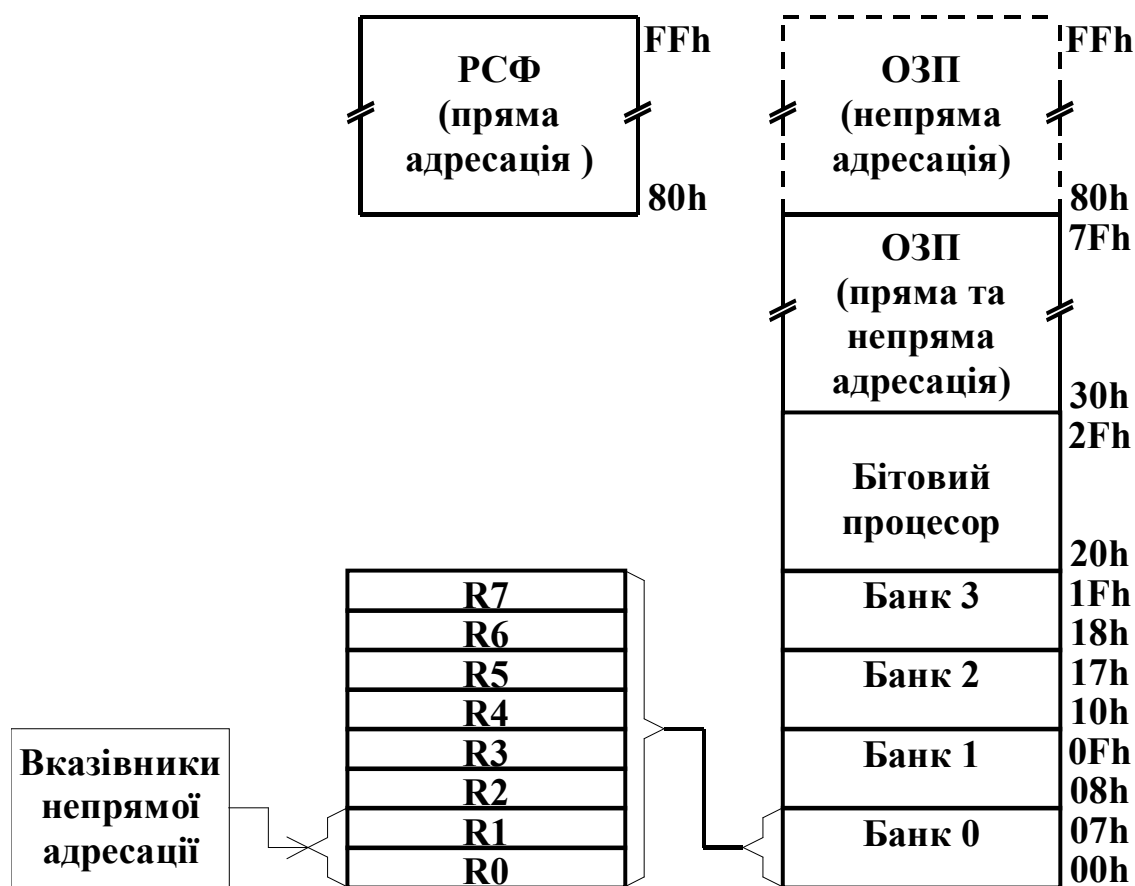


Рис. 58. Структура внутрішньої пам'яті даних

Основне ОЗП OMEOM i80C51 займає адресний простір 0...7FH окремі комірки допускають як пряму (за номером комірки), так і непряму адресацію. Регістри спеціальних функцій адресуються у просторі 80h...0FFh, але допускають лише пряму адресацію.

У модифікаціях ОМЕОМ сімейства МК51, що мають ОЗП даних ємністю 256 байтів (наприклад, i80C51FA), верхня половина ОЗП займає адресний простір 80h÷0FFh та перекриває адресний простір РСФ. Для того, аби при програмуванні відрізнити звертання до комірок верхньої половини ОЗУ від звертання до РСФ, верхня половина ОЗУ допускає лише непряму адресацію.

П'ять груп комірок пам'яті ОЗУ надають деякі додаткові можливості адресації. 32 нижні байти ОЗП утворюють 4 банки регістрів по 8 байтів кожен. Один з банків може бути визначеним поточним за допомогою встановлення відповідної комбінації бітів у регістрі слова стану PSW. Біти PSW у 4 та 3 позиції задають номер банку, що є активним. У межах поточного банку регістри допускають регістрову адресацію, а звертання до комірок пам'яті здійснюють по імені регістрів R0÷R7.

Регістри розміщуються у пам'яті у порядку збільшення їх номерів. Регістри R0 і R1 у кожному банку можуть бути використані як джерела адреси при непрямій адресації. Наприклад, якщо в комірку пам'яті з номером 9 потрібно записати число 37, то при використанні прямої адресації така дія виконується командою:

MOV A, #9

де MOV – код операції пересилання; A – назва регістра, що є адресою призначення; #9 – число 9, символ # покликаний відрізнити число від номера комірки пам'яті.

На відміну від розглянутої команди, виконання команди **MOV A, 9** скопіює у акумулятор вміст комірки пам'яті з адресою 9. Команда виконується процесором за 2 машинні цикли.

Розглянута дія може бути виконана удвічі швидше, – за 1 машинний цикл, при використанні реєстрової адресації. Якщо поточним реєстровим банком є банк 1, то комірка пам'яті з адресою 9 буде відповідати регістру R1 цього банку і запис числа в акумулятор може бути виконаний командою **MOV R1, #37**.

При використанні непрямої адресації адреса комірки пам'яті повинна бути записана в регістр R0 чи R1 поточного банку. Команда **MOV @R0, #37** завантажує у комірку ОЗП з адресою, що зберігається у R0, число 37. Використання непрямої адресації є зручним, наприклад, при виконанні дій при звертанні до масивів.

Область ОЗП, що займає 16 байтів адресного простору 20h÷2Fh, є областю напряду адресованих бітів. Будь-які з 128 бітів, що знаходяться в цій області, можуть бути використані у бітових операціях. При операціях з бітами допускається лише їх пряма адресація; непряму адресу бітів процесор не підтримує, проте програмними засобами можна виконати дії, котрі будуть виконувати встановлення, скидання та аналіз окремих бітів з непрямою адресацією.

До складу блока РСФ входять:

- ◇ побітно адресовані акумулятор АСС (адреса 0E0H) і регістр В (адреса 0F0H);
- ◇ побітно адресовані регістри портів: P0 (адреса 80H), P1 (адреса 90H), P2 (адреса 0A0H) і P3 (адреса 0B0H);
- ◇ побітно адресований регістр слова стану PSW (адреса 0D0H);
- ◇ побітно адресовані регістри керування дозволом ІЕ (адреса 0A8H) і пріоритетом ІР (адреса 0B8H) переривань;
- ◇ регістри таймерів/лічильників TH1 (адреса 8DH), TH0 (8CH), TL1 (8BH), TL0 (8AH), регістр режимів TMOD (89H) і побітно адресований регістр керування роботою TCON (88H);
- ◇ показчик даних DPTR, представлений парою регістрів - старшим DPH (адреса 83H) і молодшим DPL (адреса 82H);
- ◇ показчик стека SP (адреса 81H);
- ◇ регістр обміну SBUF (адреса 99H) і побітно адресований регістр керування SCON (адреса 98H) послідовного порту;
- ◇ регістр керування споживанням потужності PCON.

Звертання до регістрів РСФ може бути здійснене за їх абсолютними адресами, однак значно зручніше використовувати для адресації символічні власні імена регістрів РСФ, наведені вище.

При звертанні до РСФ за символічними іменами варто мати на увазі одну особливість адресації акумулятора.

Якщо акумулятор адресується як джерело або приймач байта, то для адресації використовується символічне ім'я А. Якщо ж адресуються окремі біти акумулятора, то в якості його імені використовується символ АСС.

Наприклад, записування числа 18 в акумулятор може бути здійснено командою: MOV A, #18, а встановлення в одиницю біта 7 – командою SETB ACC.7.

8. Система команд OMEOM МК-51

Мікроконтролери сімейства МК51 підтримують систему з 111 базових команд, які можна розбити на групи. Для спрощення будемо записувати позначення зображені у таблиці 35.

Таблиця 35. Умовні позначення операндів команд

Позначення	Операнд
R_n ,	Регістр поточного регістрового банку ($n = 0 \dots 7$)
R_i ,	Регістр R_0 чи R_1 поточного регістрового банку, що використовуються при непрямій адресації ($i = 0, 1$)
#d	Число довжиною 1 байт (8бітів)
#d16	Число довжиною 2 байти
ad	Пряма адреса в ОЗП даних чи в адресному просторі РСФ
bit	Адреса біта (у бітових операціях)
PC	Лічильник команд
IMEM[ad]	Уміст комірки внутрішньої пам'яті даних за адресою ad
IMEM[Ri]	Уміст комірки внутрішньої пам'яті даних, адреса якої знаходиться у R_i
EMEM[ad]	Уміст комірки зовнішньої пам'яті даних за адресою ad
PMEM[ad]	Уміст комірки пам'яті програм за адресою ad
$X := Y$	Операція пересилання даних: дані Y пересилаються у X

При виконанні арифметичних операцій операнди й результати операції подаються в стандартних для восьмирозрядних обчислювальних систем форматах, причому молодший розряд числа відповідає розряду D0, а старший –D7.

8.1. Група команд пересилання даних

Пересилання в межах внутрішньої пам'яті даних здійснюється командою MOV X, Y, де X – адреса призначення, за якою записується число чи операнд, зазначений як Y. Уміст комірки пам'яті, зазначеної як операнд Y, при виконанні операції MOV не змінюється. Припустимі варіанти пересилань у межах внутрішньої пам'яті даних зображені у вигляді графа на рис. 59.

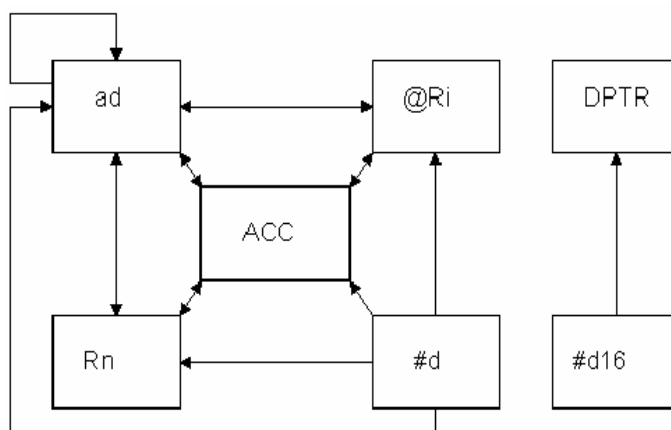


Рис. 59. Граф можливих пересилань даних

Об'єм що займається командами у пам'яті програм, швидкість виконання у машинних циклах, а також виконувані функції наведені в таблиці 36.

Таблиця 36. Команди пересилання даних у внутрішній пам'яті даних

Позначення	Кількість байтів	Кількість циклів	Операція
MOV A, Rn	1	1	A := Rn
MOV A, ad	2	1	A := IMEM[ad]
MOV A, @Ri	1	1	A := IMEM[Ri]
MOV A, #d	2	1	A := d
MOV Rn, A	1	1	Rn := A
MOV Rn, ad	2	2	Rn := IMEM[ad]
MOV Rn, #d	2	1	Rn := d
MOV ad, A	2	1	IMEM[ad] := A
MOV ad, Rn	2	2	IMEM[ad] := Rn
MOV add, ads	3	2	IMEM[add] := IMEM[ads]
MOV ad, @Ri	2	2	IMEM[ad] := IMEM[Ri]
MOV ad, #d	3	2	IMEM[ad] := d
MOV @Ri, A	1	1	IMEM[Ri] := A
MOV @Ri, ad	2	2	IMEM[Ri] := IMEM[ad]
MOV @Ri, #d	2	1	IMEM[Ri] := d
MOV DPTR, #d16	3	2	DPTR := d16

Зауважимо, що 16-розрядний регістр DPTR може адресуватись як два

восьмирозрядних регістри DPH та DPL.

Пересилання з пам'яті програм здійснюється командами MOVC (MOVE Code memory), а із зовнішньої пам'яті даних та у неї командою MOVX (MOVE eXternal memory). Множина команд MOVX та MOVC зведена у таблицю 37.

Таблиця 37. Команди роботи з зовнішньою пам'ятю даних та програм

Позначення	Операція
MOV A, @A+DPTR	$A := \text{PMEM}[A + \text{DPTR}]$
MOVC A, @A+PC	$A := \text{PMEM}[A + \text{PC}]$
MOVX A, Ri	$A := \text{EMEM}[\text{Ri}]$
MOVX A, @DPTR	$A := \text{EMEM}[\text{DPTR}]$
MOVX @Ri, A	$\text{EMEM}[\text{Ri}] := A$
MOVX @DPTR, A	$\text{EMEM}[\text{DPTR}] := A$

Команди обміну мають позначення XCH A, X. При виконанні команди вміст акумулятора обмінюється місцями із вмістом комірки пам'яті, зазначеної як операнд X. Допустимі види команд обміну і їх характеристики наведені у таблиці 38.

Таблиця 38. Команди обміну даними та роботи зі стеком

Позначення	Кількість байтів	Кількість циклів	Операція
XCH A, Rn	1	1	Обмін A з Rn
XCH A, ad	2	1	Обмін A з IMEM[ad]
XCH A, @Ri	1	1	Обмін A з IMEM[Ri]
PUSH ad	2	2	Записування даних у стек: $\text{SP} := \text{SP} + 1$, $\text{IMEM}[\text{SP}] := \text{IMEM}[\text{ad}]$
POP ad	2	2	Зчитування даних зі стека: $\text{IMEM}[\text{ad}] := \text{IMEM}[\text{SP}]$, $\text{SP} := \text{SP} - 1$

Крім зазначених команд, до групи команд передавання даних прийнято відносити операції: PUSH ad – запис у стек, POP ad – витягування зі стека. Ці команди виконуються за 2 машинних цикли і займають у пам'яті програм по 2 байти. Вказівник на вершину стека, зберігається у регістрі SP. Відмінність роботи стека OMEOM сімейства МК51 полягає в тому, що при записуванні інформації в стек спочатку інкрементується вміст SP, а потім за адресою, зазначеною в ньому, записується операнд, зазначений у команді. Витяг операнда зі стека здійснюється у зворотному порядку. Такий порядок роботи

визначає доцільність розміщення стека у верхній частині ОЗП. Природно, що розмір стека повинен враховувати не тільки об'єм інформації, яку записує за командами PUSH, але й об'єм, який записують адреси повернення при використанні переривань і підпрограм з урахуванням глибини їх вкладання та факта, що кожна адреса повернення займає 2 байти.

Слід мати на увазі, що після рестарту процесора в SP записується число 7, і, якщо в програмі використовуються переривання, підпрограми чи команди PUSH, для запобігання втрати інформації, записаної в регістри першого, другого та третього банків, необхідно змінити значення регістра SP та положення вершини стека.

8.2. Група команд арифметичних операцій

Група команд арифметичних операцій містить у собі команду додавання, віднімання, збільшення та зменшення на 1, множення та ділення, а також команда десяткової корекції. Команди додавання та віднімання можуть використовувати старе значення ознаки перенесення.

Команда	Можливі операнди	Дія
ADD A, X	ad, IMEM[ad], IMEM[@Ri], Rn	Другий операнд додається до регістра A, результат записується у A: $A := A + X$
ADDC A, X	ad, IMEM[ad], IMEM[@Ri], Rn	Другий операнд додається до регістра разом з ознакою перенесення C, результат записується у A: $A := A + X + C$
SUBB A, X	ad, IMEM[ad], IMEM[@Ri], Rn	Другий операнд віднімається від регістра разом з ознакою перенесення A, результат записується у A: $A := A - X - C$
INC X	A, IMEM[ad], Rn, IMEM[@Ri], DPTR	Збільшення на 1 комірки пам'яті чи регістра: $X := X + 1$
DEC X	A, IMEM[ad], IMEM[@Ri], Rn	Декремент комірки пам'яті чи регістра: $X := X - 1$
MUL AB	—	Беззнакове множення регістрів A та B. Результат записується на місце обох аргументів
DIV AB	—	Беззнакове ділення регістрів A та B. Результат записується на місце обох аргументів
DA A	—	Десяткова корекція акумулятора

Команда ADD реалізує функцію обчислення суми вмісту акумулятора з

операндом X. Числа мають бути без знака, або зі знаком у додатковому коді. Результат додавання записується у акумулятор, операнд X при цьому не змінюється. При виконанні команди формується ознака перенесення C.

Команда ADDC у момент обчислення суми до результату додає перенесення. У результаті виконання команди формується нове значення ознаки C, тому команда може бути використана для виконання дій над числами, довжина яких перевищує 8 розрядів.

Команда SUBB обчислює різницю операндів та від неї віднімає старе значення ознаки перенесення. Команда теж може використовуватись для виконання дій над числами, довжина яких перевищує 8 розрядів. Команда віднімання без врахування перенесення відсутня. Для виконання команди віднімання без урахування перенесення слід користуватись фрагментом:

CLR C ; скидання ознаки перенесення

SUBB A, X ; віднімання з урахуванням перенесення

Розглянемо приклад додавання трибайтових чисел. Нехай перше число розміщене в трьох комірках пам'яті за адресами 30H, 31H, 32H, причому, молодший байт розміщений в комірці 30H, а старший – у комірці з адресою 32H.

Аналогічно друге число розміщене в комірках 38H, 39H и 3AH. Суму двох чисел необхідно розмістити за місцем другого числа (рис. 60).

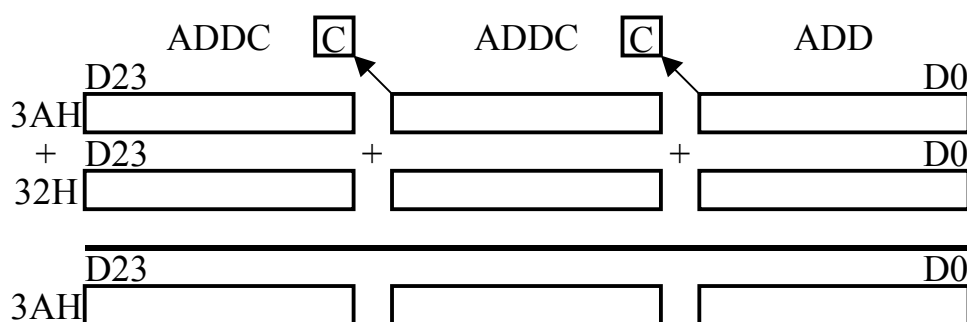


Рис. 60. Розміщення операндів багатобайтового додавання

Як і при будь-якому виді додавання, обчислення суми варто починати з молодших розрядів за допомогою команди ADD. Якщо отриманий у результаті такого додавання результат виявиться меншим, ніж максимально допустиме у рамках байта число, то перенесення з байта не виникне й ознака C = 0.

Якщо ж результат буде більшим (наприклад, якщо кожен з операндів більший за половину максимального числа), то результат не може бути представлений у межах байта; виникає перенесення $C=1$, яке варто врахувати далі при додаванні старших байтів. У загальному при додаванні старших байтів повинна бути виконана операція $X + Y + C$.

При цьому, якщо $C = 0$ (перенесення молодших байтів не виникло), буде обчислена сума у вигляді, $X + Y$, а якщо перенесення виникло, він буде врахований за всіма правилами арифметичного додавання $X + Y + 1$.

Функція $X + Y + C$ може бути виконана за допомогою команди **ADDC**, при цьому фрагмент програми, що реалізує функцію додавання двох трибайтових чисел, буде мати вигляд:

```
MOV A, 30H ;зчитування молодшого байта  
ADD A, 38H ;додавання молодших байтів  
MOV 38H,A ;збереження молодшого байта  
MOV A, 31H ;зчитування середнього байта  
ADDC A, 39H ;додавання середніх байтів та ознаки C  
MOV 39H,A ;їх збереження  
MOV A, 32H ;зчитування старшого байта  
ADDC A, 3AH ;додавання старших байтів та ознаки C  
MOV 3AH, A ;збереження результатів додавання
```

Команда **MUL AB** перемножує цілі числа без знаків з акумулятора і регістра **B**. Добуток двох восьмирозрядних чисел буде мати довжину 16 розрядів і займати 2 байти. Старший байт добутку буде поміщений у регістр **B**, а молодший – в акумулятор. Якщо результат добутку перевищує 255, то встановлюється ознака переповнення **OV** в регістрі ознак, в іншому випадку вона скидається. У будь-якому випадку скидається ознака перенесення.

Команда **DIV AB** здійснює беззнакове ділення восьмибітного цілого числа з акумулятора на восьмибітне число з регістра **B**. Частка від ділення заноситься в акумулятор, залишок – у регістр **B**. Ознаки перенесення **C** та переповнення **OV** скидаються.

Команда десятикової корекції DA A виконує дії десятикової корекції результатів додавання двійково-десятикових чисел. Команда DA A працює за наступним алгоритмом: якщо біти A_{3-0} утворюють число більше 9, або ознака AC дорівнює 1, то до регістра A додається 6, а ознака C не змінює своє значення. Далі аналізується стан чотирьох старших розрядів акумулятора. Якщо їх вміст перевищує 9 або ознака C дорівнює 1, то до A додається число 60. Основні види арифметичних операцій і їх характеристики наведено в таблиці 39.

Таблиця 39. Основні види арифметичних операцій

Позначення	Кількість байтів	Кількість циклів	Операція
ADD A, Rn	1	1	$A := A + Rn$
ADD A, ad	2	1	$A := A + IMEM[ad]$
ADD A, @Ri	1	1	$A := A + IMEM[Ri]$
ADD A, #d	2	1	$A := A + d$
ADDC A, Rn	1	1	$A := A + Rn + C$
ADDC A, ad	2	1	$A := A + IMEM[ad] + C$
ADDC A, @Ri	1	1	$A := A + IMEM[Ri] + C$
ADDC A, #d	2	1	$A := A + \#d + C$
DA A	1	1	Десяткова корекція акумулятора
SUBB A, Rn	1	1	$A := A - C - Rn$
SUBB A, ad	2	1	$A := A - C - IMEM[ad]$
SUBB A, @Ri	1	1	$A := A - C - IMEM[Ri]$
SUBB A, #d	2	1	$A := A - d - 1$
INC A	1	1	$A := A + 1$
INC Rn	1	1	$Rn := Rn + 1$
INC ad	2	1	$IMEM[ad] := IMEM[ad] + 1$
INC @Ri	1	1	$IMEM[Ri] := IMEM[Ri] + 1$
INC DPTR	1	2	$DPTR := DPTR + 1$
DEC A	1	1	$A := A - 1$
DEC Rn	1	1	$Rn := Rn - 1$
DEC ad	2	1	$IMEM[ad] := IMEM[ad] - 1$
DEC @Ri	1	1	$IMEM[Ri] := IMEM[Ri] - 1$
MUL AB	1	4	$(B, A) := A \times B$
DIV AB	1	4	$A := A \div B, B := A \bmod B$. У формулі позначення div означає дію цілочисельного ділення, позначення mod указує операцію визначення остачі від ділення, обидві дії виконуються одночасно

8.3. Група команд логічних операцій

У таблиці 40 вказані підтримувані типів логічних команд та їх опис.

Таблиця 40. Логічні команди МК-51

Позначення	Кількість байтів	Кількість циклів	Операція
ANL A, Rn	1	1	Логічне І над вмістом регістра A та регістра: $A := A \wedge Rn$
ANL A, ad	2	1	Логічне І над вмістом регістра A та комірки пам'яті або РСФ: $A := A \wedge IMEM[ad]$
ANL A, @Ri	1	1	Логічне І над вмістом регістра A та комірки пам'яті: $A := A \wedge IMEM[Ri]$
ANL A, #d	2	1	Логічне І над вмістом регістра A та константою: $A := A \wedge d$
ANL ad, A	2	1	Логічне І над вмістом регістра A та комірки пам'яті або РСФ: $IMEM[ad] := A \wedge IMEM[ad]$
ANL ad, #d	3	2	Логічне І над вмістом комірки пам'яті або РСФ та константою: $IMEM[ad] := IMEM[ad] \wedge d$
ORL A, Rn	1	1	Логічне АБО над вмістом регістра A та регістра Rn: $A := A \vee Rn$
ORL A, ad	2	1	Логічне АБО над вмістом регістра A та комірки пам'яті або РСФ: $A := A \vee IMEM[ad]$
ORL A, @Ri	1	1	Логічне АБО над вмістом регістра A та комірки пам'яті: $A := A \vee IMEM[Ri]$
ORL A, #d	2	1	Логічне АБО над вмістом регістра A та константою: $A := A \vee d$
ORL ad, A	2	1	Логічне АБО над вмістом регістра A та комірки пам'яті або РСФ: $IMEM[ad] := A \vee IMEM[ad]$
ORL ad, #d	3	2	Логічне АБО над вмістом комірки пам'яті або РСФ та константою: $IMEM[ad] := IMEM[ad] \vee d$
XRL A, Rn	1	1	Виключне АБО над вмістом регістра A та регістра Rn: $A := A \oplus Rn$
XRL A, ad	2	1	Виключне АБО над вмістом регістра A та комірки пам'яті або РСФ: $A := A \oplus IMEM[ad]$
XRL A, @Ri	1	1	Виключне АБО над вмістом регістра A та комірки пам'яті: $A := A \oplus IMEM[Ri]$
XRL A, #d	2	1	Виключне АБО над вмістом регістра A та константою: $A := A \oplus d$
XRL ad, A	2	1	Виключне АБО над вмістом регістра A та комірки пам'яті або РСФ: $IMEM[ad] := A \oplus IMEM[ad]$
XRL ad, #d	3	2	Виключне АБО над вмістом комірки пам'яті або РСФ та константою: $IMEM[ad] := IMEM[ad] \oplus d$

Група команд логічних операцій містить команди логічного АБО (ORL), І(ANL) та виключного АБО (XRL), які реалізують відповідні порозрядні операції над умістом операндів. Команди логічних операцій не змінюють регістр ознак. Для ОМЕОМ сімейства МК51 деякі логічні команди можуть не використовувати акумулятор у якості приймача даних.

8.4. Група команд операцій з бітами

ОМЕОМ сімейства МК51 підтримують групу операцій: скидання CLR, встановлення SETB, інверсію CPL, логічне І (ANL), логічне АБО (ORL) та пересилання MOV операндів, в якості яких використовуються біти – окремі розряди побітноадресованих регістрів РСФ чи біти з області прямоадресованих бітів ОЗП.

Основні характеристики і допустимі комбінації бітових операцій наведено в таблиці 41.

Таблиця 41. Команди роботи з бітами

Позначення	Кількість байтів	Кількість циклів	Операція
CLR C	1	1	Скидання біта перенесення: $C := 0$
CLR bit	2	1	Скидання біта у певній позиції $bit := 0$
SET B C			Встановлення біта перенесення: $C := 1$
SET B bit			Скидання біта у певній позиції $bit := 1$
CPL C	1	1	Інверсія біта перенесення $C := \overline{C}$
CPL bit	2	1	Інверсія біта у певній позиції $bit := \overline{bit}$
ANL C, bit	2	1	Логічне І біта перенесення та біта у певній позиції $C = C \wedge bit$
ANL C, /bit	2	1	Логічне І біта перенесення та інверсії біта у певній позиції $C := C \wedge \overline{bit}$
ORL C, bit	2	1	Логічне АБО біта перенесення та біта у певній позиції $C := C \vee bit$
ORL C, /bit	2	1	Логічне АБО біта перенесення та інверсії біта у певній позиції $C := C \vee \overline{bit}$
MOV C, bit	2	2	Перенесення біта із певної позиції у перенесення $C := bit$
MOV bit, C	2	2	Перенесення біта у певної позиції із перенесення $bit := C$

За допомогою команди SETB P1.0 можна встановити в стан логічної одиниці сигнал на виводі 1 OMEOM, а командою MOV P2.5, C вивести значення ознаки перенесення на вивід 26. Для адресації бітів використовується пряма восьмирозрядна адреса; непряма адресація бітів не підтримується.

8.5. Група команд передачі керування

Група команд передавання керування містить стандартні для будь-якого процесора команди безумовного переходу, команди умовних переходів за значеннями ознак і бітів, команди виклику та повернення з підпрограм, а також ряд оригінальних операцій, характерних лише для OMEOM сімейства МК51. Усі команди передавання керування виконуються за 2 машинних цикли. Розрізняють 2 формати команди безумовних переходів і аналогічні їм 2 формати команди виклику підпрограм: довгий – в адресному просторі 64 кбайти, та короткий виклик у адресному просторі 2 кбайти.

Команди довгого виклику та довгого переходу займають більше місця 3 байти замість двох. Ця обставина в деяких випадках може мати дуже важливе значення, оскільки будь-які умовні переходи можливі лише на відстань ± 127 байтів у внутрішній пам'яті даних відносно коду команди умовного переходу.

Умовні переходи можуть виконуватися: за значенням біта перенесення, за нульовим значенням акумулятора, за значенням біта. Існують такі команди переходів за значенням біта перенесення C: JC addr – перехід на addr, якщо C = 1, а також JNC addr – якщо C=0.

Аналіз нульового значення акумулятора проводить команда JZ addr – перейти на addr, якщо вміст акумулятора дорівнює 0 та команда JNZ addr – якщо A не дорівнює 0.

Аналіз значення біта проводять команди JB bit, addr – перехід на addr, якщо значення біта bit = 1, JNB bit, addr – перехід, якщо значення біта bit = 0 та команда JBC bit, addr – якщо значення біта bit = 1, перейти та скинути біт. Використання останньої з цих команд зручно при обслуговуванні переривань.

До команд умовних переходів відносять також команду організації циклів. Команда, що має позначення DJNZ X, addr, віднімає одиницю від вмісту операнда X, у якості якого може бути зазначений регістр чи прямоадресована комірка пам'яті, зберігає його нове значення, та порівнює результат з нулем. Якщо результат не дорівнює 0, то процесор переходить до виконання команди, розташованої за адресою addr. Якщо ж нове значення X дорівнює 0, керування передається до наступної команди. Найчастіше команда DJNZ використовується для організації циклів. Наступний фрагмент програми дозволяє організувати 36 проходів тіла циклу.

```
MOV R3, #36
```

```
M1:
```

```
...
```

```
;тіло циклу
```

```
DJNZ R3, M1
```

Команда CJNE X, #d, addr порівнює значення операнда X, у якості якого можуть бути зазначені акумулятор, чи регістр з непрямою адресою комірки пам'яті, та число #d. Якщо вони не рівні, процесор переходить до виконання команди, розташованої за адресою addr, якщо ж рівні, – до виконання команди наступної за CJNE.

Дія команди CJNE A, ad, addr аналогічна, але вона порівнює регістр A та комірку внутрішньої пам'яті даних за адресою ad. Фрагмент програми дозволяє організувати виконання циклу до тих пір, поки з порта P0 не буде зчитане значення рівне вмісту акумулятора .

```
M1:... ;тіло циклу
```

```
CJNE A, P0, STP
```

Обставина, що умовні переходи можливі тільки у області довжиною 256 байтів, часто викликає незручності при програмуванні, однак практично завжди ці незручності можуть бути обійдені.

Звертання до підпрограми можливе з будь-якого місця програми чи іншої підпрограми, при цьому процесор переходить до виконання викликаної

підпрограми, запам'ятовуючи в стеку адресу повернення. Адреса повернення займає в стеку два байти.

Повернення з підпрограми здійснюється за командою RET, розміщеною наприкінці підпрограми. Команда повернення витягає зі стека адресу повернення (два байти) та здійснює повернення за нею. Різновидом команди повернення є команда RETI. Вона використовується для повернення процесора з підпрограми переривань, її відмінності від команди RET розглядаються при описі процесу переривань. Можливі конструкції команд передавання керування і їх довжина наведені в таблиці 42.

Таблиця 42. Команди передавання керування

Позначення	Кількість байтів	Кількість циклів	Операція
1	2	3	4
LCALL ad16	3		Довгий виклик підпрограми. Адреса наступної команди записується у стек, а потім змінюється на задану у команді: $PC := PC + 3$; $SP := SP + 1$; $IMEM[SP] := PC_{7-0}$; $SP := SP + 1$; $IMEM[SP] := PC_{15-8}$; $PC := ad16$
ACALL ad11	2		Короткий виклик підпрограми. Адреса наступної команди записується у стек, а потім 11 молодших бітів змінюється на задані: $PC := PC + 3$; $SP := SP + 1$; $IMEM[SP] := PC_{7-0}$; $SP := SP + 1$; $IMEM[SP] := PC_{15-8}$; $PC_{10-0} := ad11$
LJMP ad16	3	2	Довгий перехід. Адреса змінюється задану в команді: $PC := ad16$
AJMP ad11	2	2	Короткий перехід. Адреса збільшується на 2, а потім молодші 11 бітів змінюються на ad11: $PC := ((PC + 2) \text{ AND } F800) + ad11$
SJMP rel	2	2	Відносний перехід. Адреса у PC збільшується на 2, а потім до неї додається зміщення rel, що вважається числом зі знаком: $PC := PC + 2 + rel$
JMP @A+DPTR	1		Довгий відносний перехід. Адреса збільшується на суму регістрів A та DPTR: $PC := A + DPTR$
JZ rel	2	2	Умовний перехід за значенням регістра A. Якщо A дорівнює 0, адреса у PC збільшується на 2, а потім до неї додається зміщення rel, що вважається числом зі знаком: $PC := PC + 2 + rel$, у протилежному випадку здійснюється перехід до наступної команди

Продовження таблиці 42

1	2	3	4
JNZ rel	2	2	Умовний перехід за значенням регістра A. Якщо A не дорівнює 0, адреса у PC збільшується на 2, а потім до неї додається зміщення rel, що вважається числом зі знаком: $PC := PC + 2 + rel$, у протилежному випадку здійснюється перехід до наступної команди
JC rel	2	2	Умовний перехід за перенесенням C. Якщо C дорівнює 1, адреса у PC збільшується на 2, а потім до неї додається зміщення rel, що вважається числом зі знаком: $PC := PC + 2 + rel$, у протилежному випадку здійснюється перехід до наступної команди
JNC rel	2	2	Умовний перехід за відсутності перенесення C. Якщо C дорівнює 0, адреса у PC збільшується на 2, а потім до неї додається зміщення rel, що вважається числом зі знаком: $PC := PC + 2 + rel$, у протилежному випадку здійснюється перехід до наступної команди
JB bit, rel	3	2	Умовний перехід за одиничним значенням зазначеного біта. Якщо біт дорівнює 1, адреса у PC збільшується на 2, а потім до неї додається зміщення rel, що вважається числом зі знаком: $PC := PC + 2 + rel$, у протилежному випадку здійснюється перехід до наступної команди
JNB bit, rel	3	2	Умовний перехід за нульовим значенням біта bit. Якщо біт дорівнює 0, адреса у PC збільшується на 2, а потім до неї додається зміщення rel, що вважається числом зі знаком: $PC := PC + 2 + rel$, у протилежному випадку здійснюється перехід до наступної команди
JBC bit, rel	3	2	Умовний перехід за одиничним значенням біта bit та очищення біта. Якщо біт дорівнює 1, адреса у PC збільшується на 2, а потім до неї додається зміщення rel, що вважається числом зі знаком: $PC := PC + 2 + rel$, у протилежному випадку здійснюється перехід до наступної команди. У процесі виконання команди біт скидається
DJNZ Rn, rel	2	2	Команда організації циклів з відомим числом повторень. Від регістра Rn віднімається 1, якщо новий вміст Rn не рівний 0 відбувається перехід до адреси рівної $PC + 2 + rel$, у протилежному випадку – до адреси $PC + 2$

1	2	3	4
DJNZ ad, rel	2	2	Команда організації циклів з відомим числом повторень. Від комірки пам'яті з адресою ad віднімається 1, якщо новий вміст комірки не рівний 0 відбувається перехід до адреси рівної $PC + 2 + rel$, у противному випадку – до адреси $PC + 2$
CJNE A, ad, rel	3	2	Вміст комірки пам'яті з адресою ad, порівнюється з акумулятором, якщо вміст комірки не рівний A відбувається перехід до адреси рівної $PC + 2 + rel$, у противному випадку – до адреси $PC + 2$
CJNE A, #d, rel	3	2	Вміст акумулятора порівнюється із константою d, якщо вони не рівні відбувається перехід до адреси рівної $PC + 2 + rel$, у противному випадку – до адреси $PC + 2$
CJNE Rn, #d, rel	3	2	Вміст регістра Rn порівнюється із константою d, якщо вони не рівні відбувається перехід до адреси рівної $PC + 2 + rel$, у противному випадку – до адреси $PC + 2$
CJNE @Ri, #d, rel	3	2	Вміст комірки пам'яті, адреса котрої знаходиться у Ri порівнюється з константою d, якщо вони не рівні відбувається перехід до адреси рівної $PC + 2 + rel$, у противному випадку – до адреси $PC + 2$
RET	1	2	Команда повернення із підпрограми. Вона витягає із стека адресу повернення та здійснює повернення до неї: $PC_{15-8} := IMEM[SP]$; $PC_{7-0} := IMEM[SP - 1]$; $SP := SP - 2$
RETI	1	2	Команда повернення з переривання. Виконання команди аналогічно команді RET, але додатково команда RETI розблоковує логіку переривань

8.6. Команди операцій зсуву

OMEOM сімейства МК51 підтримує лише команди операцій зсуву акумулятора без використання інших бітів та через біт перенесення у сторону старших бітів (вліво), або у бік молодших бітів (вправо).

Команда зсуву вправо без врахування біта перенесення має позначення RR A, вліво – RL A, команда зсуву вправо через біт C – RRC A, а така ж

команда зсуву вліво – RLC A.

Рух бітів у розрядах акумулятора при виконанні команд RLA і RLCA проілюстровано на рис. 61. У пам'яті програм команди зсувних операцій займають 1 байт і виконуються за 1 машинний цикл.

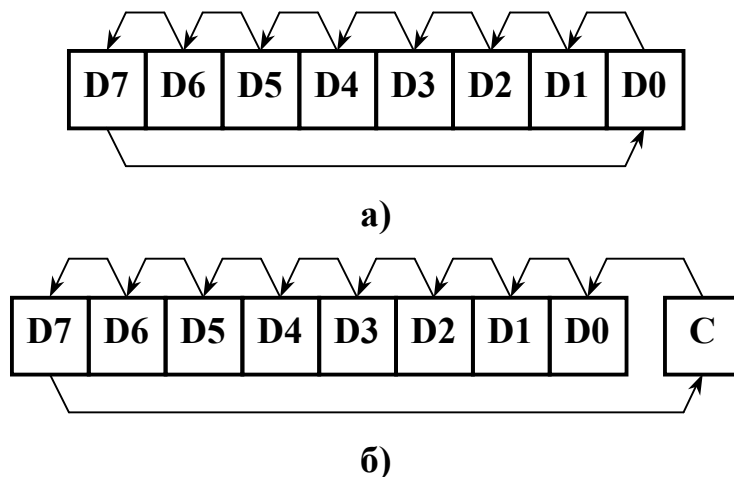


Рис. 61. Виконання команд зсувів: а) RL A, б) RLC A

8.7. Інші команди

До інших команд відносять:

- ◇ команду очищення акумулятора CLR A, виконання якої перетворює в 0 уміст акумулятора;
- ◇ команду інверсії значення акумулятора CPL A;
- ◇ "порожню" операцію NOP;
- ◇ операцію обміну тетрадами в акумуляторі SWAP A;
- ◇ операцію обміну молодшими тетрадами вмісту акумулятора і непрямоадресованої комірки пам'яті XCHD A,@Ri.

Ці команди займають у пам'яті програм по одному байту і виконуються за один машинний цикл.

9. Паралельні порти вводу-виводу інформації

Як було відзначено раніше, обмін інформацією з зовнішніми пристроями може бути здійснений через чотири восьмирозрядні паралельні порти й один послідовний. Структура портів OMEOM сімейства МК51 має ряд особливостей стосовно портів інших процесорів.

Паралельні порти P1÷P3 ОМЕОМ 80C51 мають так звану квазідвонаправлену структуру. На рис. 62 показана структура одного розряду порта P1÷P3.

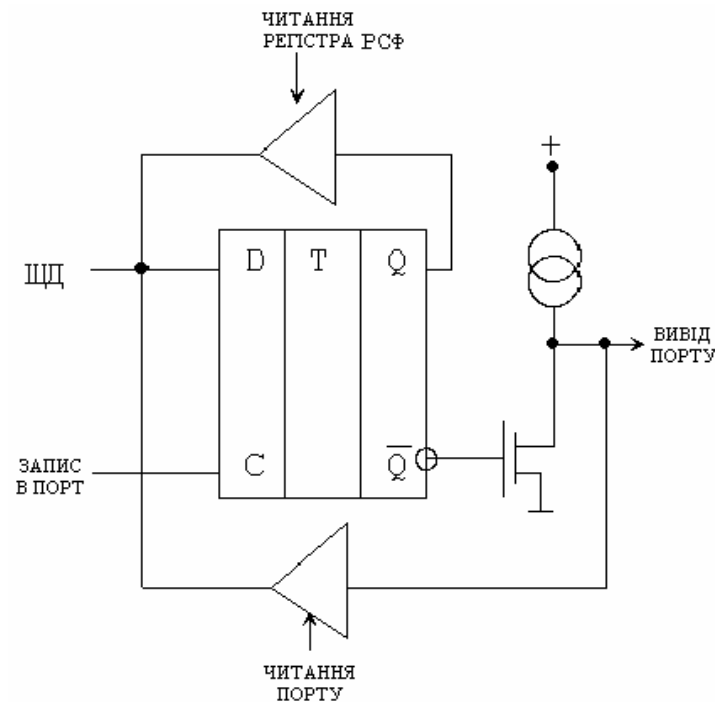


Рис. 62. Спрощена структура портів P1÷P3

При виведенні інформації через квазідвонаправлений порт із процесора на відповідний розряд шини ШД подається виведена інформація, яка супроводжується сигналом записування в порт. По фронту цього сигналу виведена інформація записується в тригер T і встановлюється на виходах Q. Якщо в тригер записується логічна одиниця, то на виході Q встановлюється потенціал логічного нуля, що замикає вихідний транзистор, у результаті чого на вивід порту надходить через схему обмеження струму від джерела живлення сигнал логічної одиниці.

Якщо ж у тригер записується логічний 0, то одиничний потенціал на виході Q введе вихідний транзистор у режим насичення, забезпечуючи замикання на "землю" виводу порту і тим самим потенціал логічного нуля на виході.

Будь-які спроби введення інформації в процесор (через нижній за схемою буфер) із зовнішнього виводу порту будуть приводити до введення логічного

нуля до тих пір, поки вихідний транзистор буде перебувати в насиченому стані. Таким чином, при роботі з портами, що мають квазідвонаправлену структуру, слід користуватись правилами:

- ◇ при виведенні через паралельні порти інформація встановлюється на відповідних виводах мікросхеми і зберігає своє значення незмінним до наступного звертання до порту;
- ◇ якщо необхідно вводити інформацію в процесор від зовнішніх пристроїв, у відповідні розряди порту до введення необхідно записати одиницю.

Записування одиниці в розряд порту закриває вихідний транзистор і в такий спосіб усуває його вплив на інформацію, що вводиться в процесор. Важливою зручністю квазідвонаправлених портів є їх здатність працювати з пасивними датчиками на взірць контактів. Пасивний датчик вмикається між виводами порту та землею схеми. Замкнений стан датчика сприймається процесором як логічний 0, а розімкнений – як логічна 1, що створюється на виводі самим портом від джерела живлення через схему обмеження вихідного струму.

Паралельний порт P0 у цілому аналогічний за структурою портам інших процесорів фірми INTEL: він має двонаправлену архітектуру, обмін інформацією з ліній порту з зовнішньою пам'яттю супроводжується стандартним набором сигналів читання (RD), записування (WR) і стробування адреси (ALE). Якщо режим звертання до зовнішньої пам'яті не використовується, робота порту P0 відбувається аналогічно портам P1...P3, за винятком того, що записування одиниць у розряди порту встановлює відповідні цим розрядам виводи OMEOM у режим холостого ходу (на відміну від портів P1...P3, на виводах яких у цьому випадку встановлюється потенціал високого логічного рівня).

При розробленні цифрових пристроїв на основі мікроконтролерів сімейства МК51 варто мати на увазі ще одну важливу особливість роботи портів.

При виконанні команд логічних операцій, інкремента, декремента, команди DJNZ, а також бітових операцій з портами в котрих регістр порту зазначений одночасно як операнд і як місце розташування результату операції, дії виконуються над умістом регістра порта, а не над комбінацією, що подається ззовні на виводи OMEOM.

10. Система переривань

Система переривань OMEOM i80C51 приймає переривання від п'яти джерел: двох зовнішніх за станом логічного сигналу на входах INT0 та INT1, двох внутрішніх за переповненням лічильника-таймера T/C0 та T/C1, а також переривання за станом послідовного порта S.

Переривання в загальному є засобом змусити процесор припинити виконання поточної програми і перейти до виконання іншої, що обслуговує пристрій, котрий запросив переривання.

Кожним джерелом може бути сформований запит на переривання, що може бути дозволене чи заборонено. Якщо обслуговування даного переривання не заборонене і процесор не зайнятий обслуговуванням переривань рівного чи вищого пріоритету, система переривань формує виклик підпрограми за адресою, фіксованою для даного переривання. А в стек буде автоматично записана адреса повернення. Адреса виклика такої підпрограми називається вектором переривання.

У ПЗП програм за адресою вектора переривання звичайно розміщується команда AJMP чи LJMP, котра передає керування на реальну адресу підпрограми обслуговування даного переривання.

Підпрограма обслуговування переривання повинна забезпечити збереження стану всіх змінних та регістрів використовуваних в основній програмі, або її підпрограмах, включаючи слово стану PSW. Підпрограма переривання повинна на початку роботи зберегти попередні значення регістрів та змінних (навіть тимчасових) основної програми, а у кінці – відновити їх, наприклад, із стека. Закінчуватися підпрограма переривань повинна командою

RETI, за якою зі стека в лічильник команд перевантажується адреса повернення, і на додаток до цього знімається блокування переривань і скидається ознака переривання, якщо вона запам'ятовувалась процесором.

Для керування перериваннями використовуються відповідні біти регістрів пріоритету IP і дозволу IE переривань.

Формат регістра дозволів переривань IE зображений на рис. 63.

D7	D6	D5	D4	D3	D2	D1	D0
EA	–	–	ES	ET1	EX1	ET0	EX0

Рис. 63. Формат регістра IE

Заборона всіх переривань здійснюється скиданням у 0 біта EA; якщо біт EA встановлений в одиницю (переривання дозволені), то дозволено обслуговування тільки тих переривань, біти дозволу яких встановлені в одиницю. Біт EX0 використовується для дозволення переривання з INT0, EX1 – для INT1, ET0 – дозволяє переривання за переповненням T/C0, ET1 – для T/C1, біт ES – для послідовного порту. Дозвіл переривання від пристрою доцільний лише у випадку його повного програмування. Якщо дозволити переривання від пристрою до встановлення його режиму можлива поява хибних переривань від пристрою. Входи переривань INT0 та INT1 не потребують вибору режиму чи програмування, тому біти EX0 і EX1 можуть бути встановлені чи скинуті програмою в будь-який момент. Біти ES, ET0 і ET1 можуть бути встановлені в одиничне значення лише тоді, коли відповідні їм пристрою повністю запрограмовані.

D7	D6	D5	D4	D3	D2	D1	D0
–	–	–	PS	PT1	PX1	PT0	PX0

Рис. 64. Формат регістра IP

Формат регістра пріоритетів переривань IP зображений на рис. 64. Кожному з джерел переривань може бути встановлений високий чи низький пріоритет установленням та скиданням відповідних бітів у регістрі IP. Біт PS задає пріоритет переривання від послідовного порта, біт PT1 – переривання від

лічильника-таймера 1, PX1 – переривання від входу INT1, PT0 – переривання від лічильника-таймера 0, PX1 – переривання від входу INT0. Підпрограми переривань вищого пріоритету можуть переривати підпрограми нижчого пріоритету. Опитування ознак здійснюється у фазі S5P2 кожного машинного такту, але виклик підпрограми переривань може відбутись лише після закінчення виконання команди. Порядок опитування ознак переривань при рівності пріоритетів наступний: INT0, T/C0, INT1, T/C1, послідовний порт.

Настроювання властивостей входів INT0 та INT1 здійснюється двома бітами регістра TCON, у регістрі TCON також відображається стан запитів переривань від входів INT0 та INT1. Формат регістра TCON зображений на рис. 65.

D7	D6	D5	D4	D3	D2	D1	D0
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Рис. 65. Формат регістра TCON

Зовнішні переривання INT0 і INT1 можуть бути запрограмовані як переривання за рівнем чи переривання за фронтом. У перериваннях за рівнем запит на переривання формується, якщо рівень логічного сигналу на вході дорівнює 0. У перериваннях за фронтом запит формується, коли сигнал на вході переходить зі стану логічної одиниці в логічний 0.

Переривання INT0 інтерпретуються процесором як переривання за спадом, якщо біт IT0 = TCON.0 встановлений в одиничний стан. При цьому запит на переривання запам'ятовується у біті IE0. Якщо переривання надходить у момент часу, коли воно заборонене чи не може бути обслуговуватися за іншою причиною, біт IE0 буде все одно встановлений. Якщо біт IE0 був встановлений, він обов'язково викличе переривання як тільки переривання із входу INT0 буде дозволено, якщо до того часу біт IE0 не буде скинутий програмно. Встановлений запит на переривання автоматично скидається при його обслуговуванні, тому немає необхідності у додаткових апаратних чи програмних засобах від повторного помилкового обслуговування переривань за фронтом. Аналогічним чином працюють біти IT0 та IE1 зв'язані з перериванням

INT1.

Режим за рівнем для входу INT0 (INT1) встановлюється скиданням у нуль біта IT0 (IT1). При цьому запит не запам'ятовується і пропадає, якщо приходить і знімається в той час, коли переривання заборонені. При використанні режиму переривань за рівнем необхідні додаткові апаратні чи програмні засоби, що забезпечують зняття чи блокування запитів на переривання після завершення його обслуговування. В іншому випадку процесор буде циклічно виконувати підпрограму обслуговування переривання, доки запит не буде знятий.

Ця властивість рівневого режиму дозволяє при використанні нескладної зовнішньої логіки легко розширити кількість входів переривань. Для розширення входу зовнішнього переривання INT0 на 8 джерел може бути використана схема рис. 66.

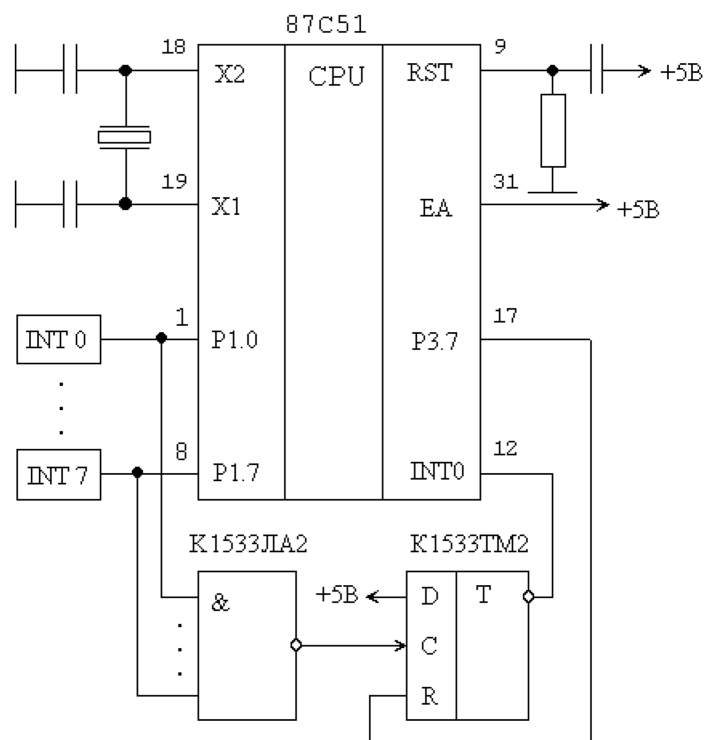


Рис. 66. Схема розширення кількості входів переривань

Якщо сигнали запитів переривань від джерел INT0...INT7 не перекриваються в часі (при необхідності така умова може бути забезпечена зовнішніми колами чи перенесенням регістра в кожну зі схем запиту з використанням додаткових ліній скидання), то при надходженні запиту від

кожного з джерел у момент переходу вихідного сигналу з логічної одиниці в логічний 0 на виході схеми "I-HE" з'явиться одиничний сигнал, по фронту якого в тригер буде записана логічна одиниця з входу D. Вона формує на виході Q потенціал логічного нуля, який буде сприйматися процесором як запит на переривання.

Коли процесор перейде до виконання підпрограми обслуговування переривання, він повинен через ввід P3.7 здійснити скидання в 0 тригера подачею логічної одиниці, що забезпечить зняття запиту з входу INT0 процесора. В програму МП системи повинні при цьому бути включені фрагменти, реалізуючі керування схемою, у програмі ініціалізації:

```
SETB P3.7 ; скидання тригера переривань
. . .      ; ініціалізація схеми, дозвіл переривань
CLR P3.7   ; дозвіл роботи схеми
у підпрограмі переривань:
ORG 3 ; за адресою 3
    AJMP INT_PR ; перейти до початку підпрограми
. . .
INT_PR:
    SETB P3.7 ; сформувати імпульс
    CLR P3.7  ; скидання тригера
    PUSH PSW  ; зберегти ССП
    JNB P1.0, INT_0 ; обслуговування переривання INT 0
    JNB P1.1, INT_1 ; обслуговування переривання INT 1
    . . .
    JNB P1.7, INT_7 ; обслуговування переривання INT 7
INT_END:
    POP PSW відновлення регістра ознак
    RETI повернення до основної програми
INT_0:
    ; команди обслуговування переривання за INT 0
```

JMP INT_END; повернення до загальної підпрограми

Схему можна спростити використавши замість зовнішнього тригера інвертор та внутрішній тригер дозволення переривання EX0.

Переривання від лічильників-таймерів працює за фронтом і не вимагають програмного скидання ознаки при обслуговуванні. Стан біта переривання лічильника-таймера відображений у регістрі TCON (рис. 65) у бітах TF0 та TF1 для лічильника-таймера 0 та 1, відповідно.

Послідовний порт формує єдиний запит переривання при двох різних подіях: прийманні чергового байта в внутрішній буфер та закінченню передавання байта із буфера передавача. Паралельно із запитом переривання формуються дві ознаки "передавач порожній" TI та "приймач повний" RI, що можуть бути використані для ідентифікації події, пов'язаної із запитом переривання. Ознаки запитів не скидаються автоматично при обслуговуванні переривання та вимагають програмного скидання, тому найдоцільнішим є ідентифікація події за допомогою команди JBC:

ORG 23h

**AJMP SER_P ;перехід від вектора переривання
;до підпрограми його обслуговування**

SER_P:

PUSH PSW

JBC RI, INPUT ;ідентифікація закінчення прийому

JBC TI, OUTPUT ;ідентифікація закінчення передачі

INPUT: ;код обробки прийнятих даних

. . .

AJMP END

OUTPUT: ;код видавання нових даних

. . .

END:

POP PSW

RET I

Використання команд JBC дозволяє ідентифікувати подію. Наприклад, перейти до програми обслуговування приймання сигналів у послідовний порт, якщо ознака прийому RI встановлена у 1 та одночасно здійснити програмне скидання встановленої ознаки.

Таблиця 43. Регістри керування та стану переривань

Переривання	Умова	Ознака наявності	Вектор	Дозвіл	Пріоритет
INT0	0 або перехід 1/0	IE0=TCON.1	0003H	EX0=IE.0	PX0=IP.0
INT1	0 або перехід 1/0	IE1=TCON.3	0013H	EX1=IE.2	PX1=IP.2
T/C0	Переповнення T/C0	TF0=TCON.5	000BH	ET0=EX.1	PT0=IP.1
T/C1	Переповнення T/C0	TF1=TCON.7	001BH	ET1=EX.3	PT1=IP.3
Послідовний порт	Приймач повний або передавач порожній	RI=SCON.0 TI=SCON.1	0023H	ES=IE.4	PS=IP.4
Всі	—	—	—	EA=IE.7	—
Значення бітів	—	1 – наявність 0 – відсутність	—	1 – дозвіл, 0 – заборона	1 – високий, 0 – низький

11. Лічильники-таймери

В архітектуру OMEOM i80C51 входять два ідентичних шістнадцятирозрядних лічильники-таймери T/C0 та T/C1, кожний з яких може бути використаний або в якості таймера, або як лічильник зовнішніх подій.

При роботі як таймер уміст T/C0 та T/C1 інкрементується в кожному машинному циклі, тобто через кожні 12 періодів тактового сигналу. При роботі як лічильник уміст T/C0 та T/C1 інкрементується під впливом переходу з 1 у 0 зовнішнього вхідного сигналу, що подається на відповідний вивід МК-51. Опитування значення зовнішнього вхідного сигналу виконується в момент часу S5P2 кожного машинного циклу.

Вміст лічильника буде збільшений на 1 у випадку, якщо в попередньому циклі був зчитаний вхідний сигнал високого рівня (1), а в наступному – сигнал низького рівня (0). Нове інкрементоване значення лічильника буде сформоване у момент S3P1 у циклі, що настає за тим, у якому був виявлений перехід з 1 в 0. На розпізнавання переходу потрібно два машинних цикли, тому максимальна частота підрахунку вхідних сигналів дорівнює 1/24 частоти резонатора. Лічильник відображається на два восьмирозрядних регістра спеціальних

функцій. Зазвичай регістр TH зберігає старший байт лічильника, TL – молодший. Кожен лічильник може бути запрограмований для роботи в одному з чотирьох режимів роботи.

D7	D6	D5	D4	D3	D2	D1	D0
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Рис. 67. Формат регістра TCON

Вмикання лічильників, а також стан переповнення може бути перевірений через регістр TCON (рис. 67). Встановлення біта TR0 у регістрі TCON запускає лічильник-таймер T/C0, а біт TR1 – T/C1. Біт TF0 вказує на переповнення лічильника-таймера T/C0. Встановлюється при переході лічильника із стану 0FFFFh (0FFh) у стан 0. Очищується при передаванні керування на процедуру обробки переривання, або програмним шляхом. Біт TF1 виконує такі ж функції для лічильника-таймера 1.

Для завдання режимів роботи таймерів використовується доступний лише побайтово регістр TMOD із регістрів спеціальних функцій, а для оперативного керування його роботою – побітноадресований регістр TCON.

D7	D6	D5	D4	D3	D2	D1	D0
GATE 1	CT1	M1.1	M1.0	GATE 0	CT0	M0.1	M0.0
Лічильник-таймер 1				Лічильник-таймер 0			

Рис. 68. Формат регістра TMOD

Формат регістра TMOD зображений на рис. 68. Біт керування таймером GATE0 керує вмиканням лічильника-таймера. Коли GATE0 = 1 лічильник-таймер 0 працює завжди коли біт TR0 = 1, якщо ж GATE0 = 1 – лічильник-таймер працює лише якщо TR0 = 1 та INT0 = 1. Біт CT0 задає тип подій для лічильника-таймера 0, якщо CT0 рівний нулю лічильник працює у режимі лічильника подій, якщо одиниці – у режимі таймера. Один із чотирьох режимів T/C формується двійковою комбінацією бітів M0.1 та M0.0. Біти лічильника-таймера 1 мають аналогічне призначення.

У всіх режимах роботи, крім режима 3 вхідна схема лічильника-таймера 0

має структуру зображену на рис. 69. Структура лічильника-таймера 1 аналогічна та відрізняється використанням інших керуючих входів та бітів, а також іншою назвою біта стана.

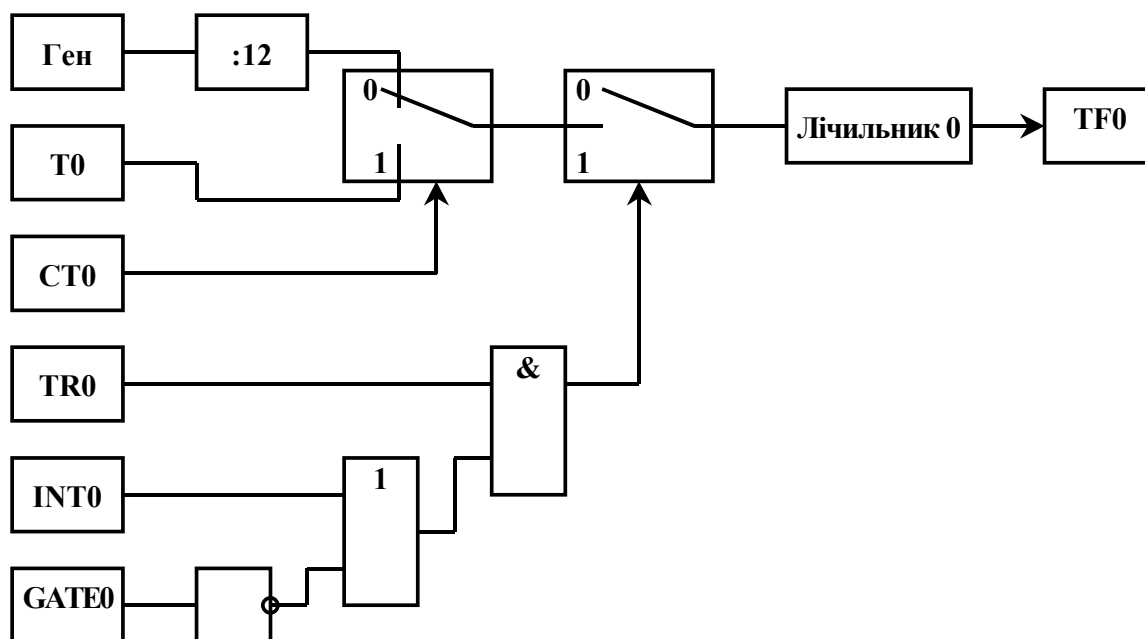


Рис. 69. Узагальнена структура лічильника-таймера у режимах 0÷2

Розглянемо режими роботи лічильника-таймера. Режим 0 призначений для емуляції лічильника-таймера МК-48. В цьому режимі регістр ТНх працює як 8-розрядний лічильник з можливістю встановлення початкового значення, а лічильник ТLх – як 5-розрядний попередній подільник.

У режимі 1 регістри ТНх та ТLх працюють як 16-розрядний лічильник з можливістю встановлення початкового значення. Робота таймерів / лічильників у режимах 0 і 1 ідентична за винятком того, що в режимі 0 ТL0 (ТL1) працює в режимі п'ятирозрядного лічильника (з метою сумісності з попереднім поколінням мікропроцесорів сімейства MCS 48), а в режимі 1 – восьмирозрядного.

Для використання лічильників-таймерів як лічильників зовнішніх подій біт СТ0 (СТ1) = ТМ0Д.2 (ТМ0Д.6) повинен бути встановлений в одиницю, при цьому вхід лічильника ТL0 (ТL1) під'єднується до зовнішнього виводу ОМЕОМ Т0 (Т1). Для використання в якості таймерів цей біт повинен бути скинутий у 0, при цьому на вхід ТL0 (ТL1) від задаючого генератора, подаються імпульси з

періодом машинного циклу.

Робота лічильника-таймера може бути в будь-який момент зупинена програмно скиданням біта $TR0 = TCON.4$ ($TR1 = TCON.6$). Якщо біт $TR0$ ($TR1$) встановлений в одиницю, а біт $GT0 = TMOD.3$ ($GT1 = TMOD.7$) скинутий в 0, то лічильник-таймерів працює безупинно. Якщо ж біти $TR0$ ($TR1$) і $GT0$ ($GT1$) встановлені в одиничне значення, то робота лічильника-таймера зупиняється при подаванні потенціалу логічного нуля на вхід $INT0$ ($INT1$). У такий спосіб може бути реалізоване зовнішнє керування лічильником-таймером.

У режимі 2 лічильники-таймери працюють з автоперезавантаженням. У цьому режимі для відліку використовується лише восьмирозрядний лічильник $TL0$ ($TL1$), який в момент переходу зі стану FFH завантажується вмістом регістра $TH0$ ($TH1$). Режим 2 використовується, як правило, для організації циклічних процесів із фіксованим значенням часу циклу. Наприклад у режимі 2 зручно формувати мітки реального часу та сигнали синхронізації послідовного порта.

У режимі 3 може працювати лише лічильник-таймер $T/C0$, встановлення режиму 3 для лічильника-таймера $T/C1$ призводить до його зупинки зі збереженням вмісту $TH1$ і $TL1$. При встановленні у режим 3 лічильника-таймера $T/C0$ вхідні кола обох лічильників приймають конфігурацію зображену на рис. 70.

Лічильник-таймер $T/C0$ розділяється на два незалежні канали $TH0$ та $TL0$, кожен зі своїм керуванням та джерелом сигналу. При роботі у режимі 3 лічильник-таймер $T/C0$ позбавляє лічильник-таймер $T/C1$ кола керування за сигналом $TR1$ та можливості формування запитів переривань. Отримані незалежні канали $TH0$ і $TL0$ функціонують два восьмибітних пристрої, один з яких ($TH0$) може використовуватися лише як таймер, а інший ($TL0$) – і як таймер і в якості лічильника зовнішніх подій з керуванням, аналогічним режимам 0, 1 і 2.

При встановленні $T/C0$ в режим 3 лічильник-таймер $T/C1$ може виконувати будь-які функції, що не потребують формування запиту на

переривання. Одним з таких використання є керування швидкістю роботи послідовного порту.

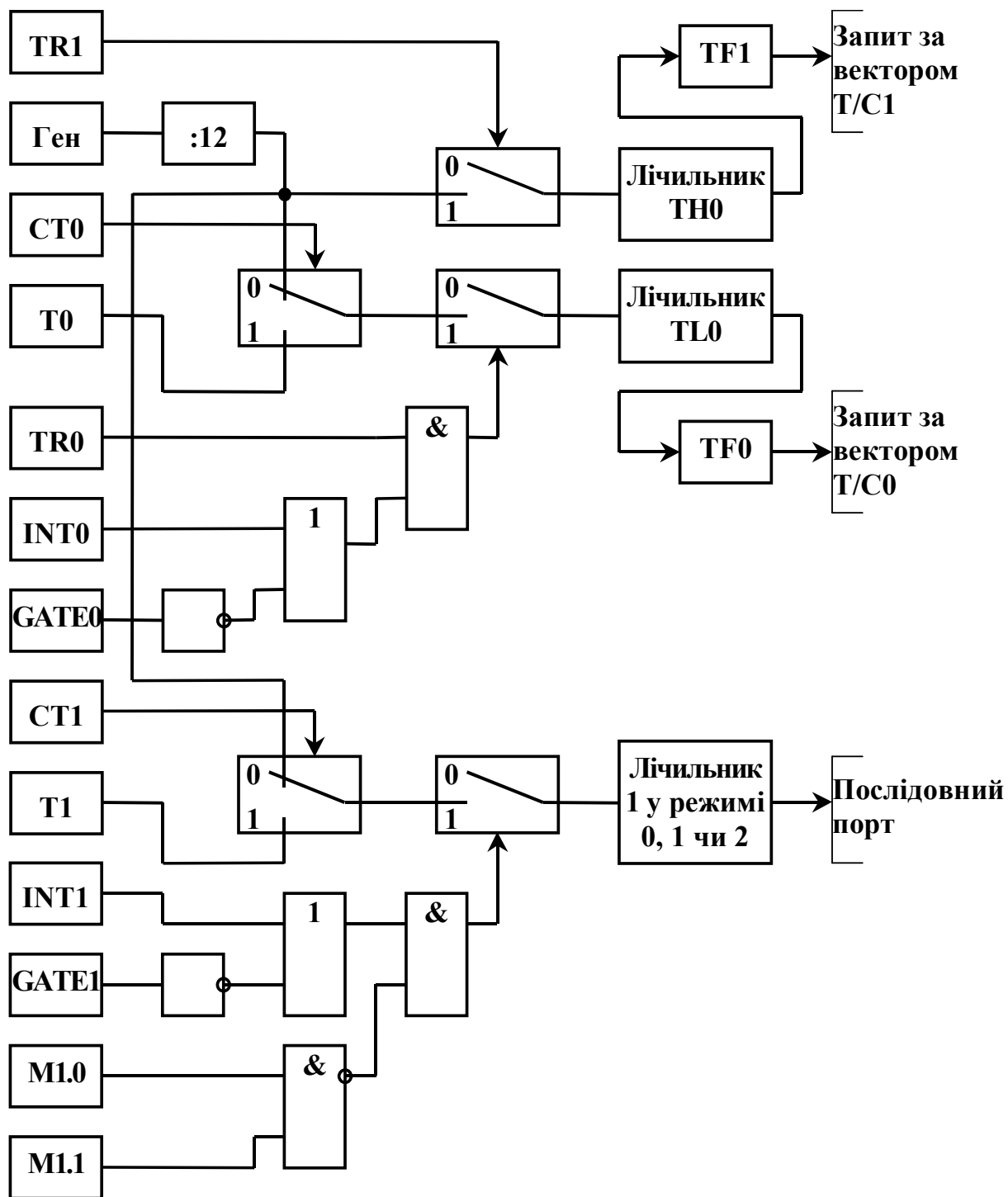


Рис. 70. Узагальнена структура лічильників-таймерів у режимі 3 T/C0

Лічильник-таймер 1 при встановленні T/C0 у режим 3 може бути переведений у кожен з режимів 0, 1 чи 2, однак втрата кола керування по біту TR1 позбавляє лічильник-таймер 1 можливості гарантованої зупинки за цим колом. Однак зупинка лічильника-таймера 1 може бути проведена шляхом

переведення його у режим 3. Отже зупинка лічильника-таймера 1 може бути здійснена за командою ORL TMOD, #00110000B.

12. Послідовний порт

Послідовний порт OMEOM сімейства МК-51 забезпечує повний дуплексний режим обміну інформацією, представленою послідовним кодом. Обмін даними здійснюється молодшими бітами вперед.

До складу послідовного порту входять:

- ◇ реєстри зсуву,
- ◇ передавання та приймання,
- ◇ буферний реєстр SBUF прийомопередавача, що входить до складу реєстрів спеціальних функцій (PCФ).

Для керування роботою послідовного порту використовується побітно-адресований реєстр PCФ SCON, а також біт SMOD у реєстрі PCON.

D7	D6	D5	D4	D3	D2	D1	D0
SMOD	–	–	–	GF1	GF0	PD	IDL

Рис. 71. Формат реєстра PCON

Формат реєстра PCON зображений на рис. 71. Біт SMOD керує швидкістю роботи послідовного порта. При виборі SMOD = 1 швидкість передавання даних у деяких режимах подвоюється.

D7	D6	D5	D4	D3	D2	D1	D0
SM0	SM1	SM2	REN	TB8	RB8	TI	RI

Рис. 72. Формат реєстра SCON

Призначення бітів реєстра SCON зображено на рис. 72.

Біти SM0÷SM1 задають режим роботи послідовного порта. Послідовний порт може працювати в одному з чотирьох режимів роботи. Режим 0 є синхронним, а решта – асинхронними. Стан біта SM2 змінює ознаки режиму роботи. Його використання залежить від режиму роботи. Біт REN у стані одиниці дозволяє приймання даних. Біти TB8 та RB8 визначають стан дев'ятого біта при передаванні та прийманні даних. Встановлення TI у одиницю вказує на

запит переривань від передавача, а RI – від приймача.

Режим 0 встановлюється комбінацією бітів $SM0 = 0$ та $SM1 = 0$. У цьому режимі приймання та передавання інформації здійснюються через вивід Rx (P3.0). Вивід Tx (P3.1) є виходом, по якому з процесора видаються імпульси зсуву, що супроводжують дані. При виведенні інформації з МК-51 імпульси зсуву формуються процесором так, що їх спади (переходи з 1 у 0) відповідали середині переданого символу. Спади Tx можуть бути використані для фіксації виведених даних.

При введенні інформації в контролер (прийманні) її читання з виводу Rx здійснюється у фазі S5P2 машинного циклу; імпульси зсуву встановлюються в одиницю у фазі S6P1, а в 0 – у фазі S3P1. Таким чином, при введенні інформації в процесор через послідовний порт у режимі 0 повинна бути забезпечена її синхронність відносно генерованих процесором імпульсів зсуву, причому зміна символу, що вводиться, може здійснюватися як по фронту (перехід з 1 у 0), так і по спаду імпульсів зсуву. Швидкість обміну в режимі 0 фіксована та складає $1/12$ тактової частоти

Передавання з послідовного порту починається автоматично при виконанні процесором будь-якої команди, що реалізує записування інформації в регістр SBUF. Після того, як уміст SBUF буде передано на вивід Rx, автоматично встановлюється ознака TI та формується запит на переривання.

Приймання інформації через послідовний порт починається при одночасному виконанні умов $REN = 1$ та $RI = 0$. Приймання даних дозволено, а дані з попереднього приймання даних зчитані. При виконанні цих умов послідовний порт на виході Tx створює вісім імпульсів зсуву, заповнюючи зсувний регістр інформацією з виводу Rx, після чого встановлює ознаку RI у стан 1 та формує запит на переривання. Зауважимо, що пасивний стан лінії Tx – високий.

Режим 1 встановлюється комбінацією бітів $SM0 = 0$ та $SM1 = 1$. У цьому режимі роботи обмін здійснюється у асинхронному режимі: передавання інформації через вивід Tx, а приймання – через Rx.

З метою синхронізації передаванні дані доповнюється двома службовими бітами: стартовим та стоповим. Першим завжди передається стартовий біт , що рівний 0, далі передаються 8 бітів даних, а у кінці – стоп-біт завжди рівний 1 (рис. 73). Службові біти одночасно використовують для поділу переданих байтів. При відсутності обміну на лініях порту встановлюється високий логічний рівень.

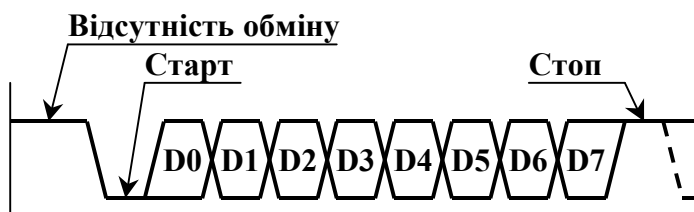


Рис. 73. Формат даних у асинхронних режимах роботи

Передавання починається при записуванні інформації в SBUF. Інформація разом із службовими бітами передається МК-51 за допомогою лінії Tx.

Поява рівня логічного нуля на вході Rx сприймається портом як початок передавання байта та при виконанні умови $REN = 1$ та $RI = 0$ ініціює початок приймання стартового символу. Приймання кожного символу здійснюється трикратним опитуванням із мажоритарним принципом: вважається, що прийнято одиницю, якщо хоча б у двох випадках опитування лінії із трьох ліній перебувала в одиничному стані.

Якщо стартовий символ прийнятий, приймаються ще 9 символів: вісім інформаційних бітів та стоп-біт, що записується позицію біт RB8 регістра SCON. Далі встановлюється ознака RI 1 у стан 1 та формується запит на переривання.

Якщо ж стартовий символ не сприйнятий, порт повертається у початкове положення чекання старт-біта.

З метою запобігання помилкового приймання послідовним портом повідомлень, формат яких не відповідає зображеному на рис. 73, передбачено спеціальний режим захисту від помилок формату, встановлює одиничне значення біта SM2.

Якщо встановити цей біт в одиницю, то послідовний порт буде аналізувати сигнал на позиції стоп-біта, і якщо цей сигнал буде прийнятий як 0, біт RI не буде встановлено у 1, та не буде сформовано запит на переривання, а прийнята інформація буде відкинута.

Швидкість передавання у режимі 1 визначається частотою переповнення лічильника-таймера 1, який може при цьому працювати як таймер чи як лічильник зовнішніх подій у кожному з режимів 0,1 чи 2. Тому швидкість передавання даних може змінюватись в широких межах. У загальному вигляді швидкість передавання можна описати формулою

$$F_N = \frac{2^{SMOD}}{32} F_{OVT1}$$

де F_{OVT1} – частота переповнення лічильника-таймера T/C1; SMOD – значення біта SMOD регістра PCON.

Режим 2 встановлюється комбінацією бітів SM0 = 1, SM1 = 0. Формат переданого слова в режимі 2 зображений на рис. 74. Біт TB8 програмно встановлюється у регістрі спеціальних функцій SCON.

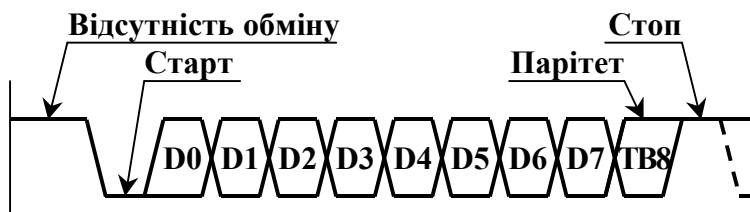


Рис. 74. Формат даних у режимі 2 та 3

Дії послідовного порта у режимі 2 в основному аналогічні діям у режимі 1. Швидкість передавання та приймання даних у режимі 2 фіксована і складає

$$F_N = \frac{2^{SMOD}}{64} F_T$$

де F_T – тактова частота задаючого генератора.

Передавання додаткового символу TB8, прийнятого на протилежному боці у тригер RB8, надає додаткові можливості послідовного обміну. Наприклад, при обміні по лінії з відчутним рівнем перешкод символ TB8 може бути використаний як символ парності для перевірки наявності помилок. Крім

того, цей символ зазвичай використовують при побудові систем з кодово-адресним принципом опитування об'єктів, що підключені до ведучого контролера одним колом.

У цьому випадку звертання до окремих об'єктів з боку керуючого контролера або комп'ютера здійснюється в два етапи: спочатку передається адреса пристрою, а потім інформація призначена для нього. Біт TV8 при цьому може бути ознакою характеру інформації, що передається, та використовуватися кожним з паралельно під'єднаних до контролера пристроїв для ідентифікації належності до нього інформації, переданої в даному сеансі.

Режим 3 встановлюється комбінацією біт $SM0 = 1$, $SM1 = 1$. У цьому режимі робота послідовного порту аналогічна роботі в режимі 2, а швидкість передавання та приймання даних така ж як і у режимі 1.

13. Режим зниженого енергоспоживання

У мікроконтролерах сімейства МК-51, що виконані по КМОН технології існують два режими зниженого енергоспоживання. Режим зниженого енергоспоживання задається регістром PCON.

D7	D6	D5	D4	D3	D2	D1	D0
SMOD	–	–	–	GF1	GF0	PD	IDL

Рис. 75. Формат регістра PCON

Формат регістра PCON зображений на рис. 75. Біт SMOD керує швидкістю роботи послідовного порта. При виборі $SMOD = 1$ швидкість передавання даних у деяких режимах подвоюється.

Біти GF1 та GF0 є бітами загального призначення. Вони призначені для збереження бітів користувача.

Біт керуванням споживанням PD вмикає одиничним значенням режим зниженого енергоспоживання. Біт режиму холостого ходу IDL вмикає одиничним значенням режим холостого ходу. Якщо встановлені обидва біти (PD та IDL) вмикається режим мікроспоживання.

У режимі холостого ходу центральний процесор відмикається, а система

переривань, лічильники-таймери та інші блоки продовжують функціонування. Лічильник команд, регістри та внутрішнє ОЗП даних зберігає свої значення. Споживана потужність у режимі холостого ходу становить коло 15% від номінальної.

Вихід із холостого ходу здійснюється за сигналом переривання, або за сигналом RESET. Прийняте переривання автоматично скидає біт IDL.

У режимі мікроспоживання зупиняється виконання всіх функцій мікроконтролера, регістри спеціальних функцій втрачають своє значення, лишається незмінним лише вміст ОЗП. Вихід зі стану мікроспоживання можливий лише за сигналом RESET.

Контрольні запитання

1. Однокристальні мікро-ЕОМ сімейства MCS51. Основні характеристики.
2. Архітектура ОМЕОМ 87С51.
3. Будова арифметико-логічного пристрою.
4. Організація резидентної пам'яті даних та пам'ять програм.
5. Система переривань однокристальних мікро-ЕОМ сімейства MCS51.
6. Порти вводу-виводу.
7. Послідовний інтерфейс.
8. Таймери/ лічильники. Режим роботи.
9. Під'єднання до МК51 зовнішньої пам'яті даних та програм.

Мікроконтролери PIC16C5X та PIC16C8X

Всі мікроконтролери сімейства PIC16 та PIC17 побудовані на основі гарвардської архітектури – концепції роздільних шин областей пам'яті для даних та команд. Така концепція забезпечує просту, але потужну систему команд, розроблену так, що бітові, байтові і реєстрові операції працюють з високою швидкістю і з перекриттям за часом вибірок команд і циклів виконання.

Шина даних і пам'ять даних (ОЗП) мають ширину 8 бітів, а програмна шина і пам'ять програм (ПЗП) – ширину 12 бітів. 12-бітова ширина програмної пам'яті забезпечує вибірку 12-бітової команди за один цикл. У PIC16 та PIC17 програмна пам'ять розташована всередині кристала. Програма, що виконується, може зберігатися лише у вбудованому ПЗП.

Двоступінчастий конвеєр забезпечує одночасну вибірку і виконання команд, завдяки чому всі команди виконуються за один цикл, крім команд переходів. У таблиці 44 наведені основні характеристики мікроконтролерів групи PIC16C5XX.

Таблиця 44. Основні характеристики мікро-ЕОМ сімейства PIC16C5xx

Назва	Частота, МГц	Кількість виводів	Кількість портів	ПЗП	ОЗП
PIC16C52	4	18	12	384x12 ППЗП	25x8
PIC16C54	20	18	12	512x12 ППЗП	25x8
PIC16C54A	20	18	12	512x12 ППЗП	25x8
PIC16CR54	20	18	12	512x12 ПЗП	25x8
PIC16CR54A	20	18	12	512x12 ПЗП	25x8
PIC16CR54B	20	18	12	512x12 ПЗП	25x8
PIC16C55	20	28	20	512x12 ППЗП	25x8
PIC16C56	20	18	12	1024x12 ППЗП	25x8
PIC16CR56	20	18	12	1024x12 ПЗП	25x8
PIC16C57	20	28	20	2048x12 ППЗП	72x8
PIC16CR57A	20	28	20	2048x12 ПЗП	72x8
PIC16CR57B	20	28	20	2048x12 ПЗП	72x8
PIC16C58A	20	18	12	2048x12 ППЗП	72x8
PIC16CR58A	20	18	12	2048x12 ПЗП	72x8
PIC16CR58B	20	18	12	2048x12 ПЗП	72x8

Система команд складається із 33 команд, котрі виконуються за один цикл (крім команд переходу). Кожна команда займає у пам'яті 12 біт та призначена для виконання дій над 8-бітними даними з використанням прямої, безпосередньої, непрямої та відносної адресації даних та команд.

Статичний принцип в архітектурі та економічна високошвидкісна КМОН технологія виготовлення призводять до широкого діапазону робочої частоти при невеликому струмі споживання. Робоча тактова частота мікроконтролера може лежати у межах від 0 Гц до 20 МГц. Завдяки КМОН технології виготовлення у мікроконтролерах серії PIC16C5X досягнутий широкий діапазон напруги живлення, котра може становити від 2 до 6 В. Струм споживання становить 2 мА при живленні від 5 В та тактовій частоті 4 МГц, 15 мкА для живлення від 3 В при тактовій частоті 32 кГц та 1 мкА у режимі чекання при 3 В. У випадку необхідності, режим економії SLEEP дозволяє суттєво знизити енергоспоживання мікроконтролера. Разом з тим досягнута можливість отримання струму в 10 мА з виходів мікроконтролера. Діапазон робочих температур у комерційному виконанні становить від 0 до +70 °С, у промисловому – від -40 до +70 °С.

Мікроконтролери мають 12 або 20 ліній вводу-виводу з індивідуальним налаштуванням кожної з них, Watchdog-таймер (WDT) із власним убудованим генератором, що забезпечує підвищену надійність, 8-бітний лічильник-таймер RTCC з 8-бітним програмованим попереднім подільником та таймер ввімкнення при скиданні. Спеціальна схема у мікроконтролері дозволяє автоматичне скидання при ввімкненні живлення мікросхеми без використання додаткових елементів.

Вбудований генератор має біти для вибору конфігурації для роботи у режимі: RC-генератора (RC), звичайного кварцового резонатора (XT), високочастотного кварцового резонатора (HS), економічного низькочастотного резонатора (LP).

Пам'ять мікроконтролера поділяється на ОЗП даних, ПЗП програм, дворівневий апаратний стек та EPROM пам'ять даних. Також у

мікроконтролерах виділяють 7 або 8 спеціальних апаратних регістрів SFR. EPROM має біт таємності для захисту коду, біти ідентифікації, тощо. У OMEOM є вбудований пристрій для програмування пам'яті програм, що дозволяє використовувати при програмуванні лише два виводи.

Усю оперативну пам'ять даних поділяють на дві частини: спеціальні регістри та регістри загального призначення. Всі спеціальні регістри та лічильник команд відображаються на пам'ять даних. До складу спеціальних регістрів входять: регістр лічильника-таймера реального часу (TMR0), лічильник команд (PC), регістр стану (STATUS), регістри вводу-виводу (PORT) та регістр непрямої адресації (FSR). Крім того, спеціальні регістри керують конфігурацією портів вводу-виводу і режимів попереднього подільника. Регістри загального призначення використовуються програмою для збереження змінних. Адресація всіх регістрів і комірок пам'яті у мікроконтролерах сімейства PIC16C5X може бути прямою та непрямою.

Мікроконтролери PIC16C5X мають ортогональну (симетричну) систему команд, що дозволяє виконувати будь-яку операцію з будь-яким регістром, використовуючи будь-який метод адресації. Це полегшує програмування для них і значно зменшує час, необхідний на навчання роботи з ними.

У мікроконтролерах PIC16C5X є 8-розрядний арифметико-логічний пристрій (АЛП) та пов'язаний із ним робочий регістр W. АЛП виконує додавання, віднімання, зсув, бітові і логічні операції. В командах, які мають два операнди, одним із них є робочий регістр W, другий операнд може бути константою чи вмістом будь-якого регістра чи комірки ОЗП. Робочий регістр W знаходиться за межами адресного простору ОЗП. Визначити його адресу або звернутись за непрямою адресою до нього неможливо.

У командах із одним операндом операнд може бути вмістом робочого регістра, вмістом будь-якого регістра або комірки пам'яті даних. Для виконання всіх операцій АЛП використовує робочий регістр W. Залежно від результату виконання операції можуть змінитися значення бітів перенесення C, десяткового перенесення DC і нуля Z в регістрі стану STATUS. При відніманні

біти C і DC працюють як біти позичання та десяткового позичання. Регістр STATUS доступний для будь-якої команди так само, як і будь-який інший регістр. До його складу входять біти TO і PD, які встановлюються апаратно і не можуть бути змінені програмно. Дані розряди встановлюються у відповідні стани при ввімкненні живлення, скиданні і переході в режим SLEEP. Виконуючи їх програмне опитування, можна визначити спосіб запуску програми.

Сімейство мікроконтролерів PIC16C5X містить дворівневий апаратний стек. При виконанні команди звертання до підпрограми у вершину стека завантажується лічильник команд, попередньо збільшений на одиницю. Одночасно старе значення з вершини стека копіюється у другий рівень стека.

Одною із основних відмінностей портів вводу-виводу є те, що програма може зчитувати і записувати дані в них аналогічно регістрам загального призначення. При читанні завжди зчитується дійсний стан виводів незалежно від того, запрограмовані окремі розряди як входи чи як виходи. Після скидання всі розряди програмуються як входи. На виходах портів є тригери і їх стан не змінюється до наступного записування в порт. Установка режиму кожного розряду у всіх портах виконується за допомогою установки відповідних розрядів у регістрах керування режимами портів TRIS (TRISA, TRISB чи TRISC).

Модуль таймера (TMR0) у даних мікроконтролерах має у собі 8-розрядний лічильник-таймер, доступний для читання і записування даних, 8-розрядний програмований попередній подільник, який може бути програмно під'єднаний до таймера чи до WDT. При програмуванні лічильника-таймера задають внутрішнє чи зовнішнє джерело тактових сигналів, у випадку зовнішнього джерела є можливість обрання фронту тактуючого імпульсу. Таймер має два режими роботи: режим таймера і режим лічильника. В режимі таймера вміст TMR0 збільшується в кожному командному циклі (якщо немає попереднього подільника). В режимі лічильника вміст TMR0 збільшується по кожному перепаду 1/0 чи 0/1 на виводі TOCK1. Перепад, який збільшує

значення TMR0, вибирається установкою відповідного біта у службовому регістрі OPTION, що програмно доступний для записування даних. Цим ж регістром виконується встановлення режиму роботи попереднього подільника (TMR0/WDT) та значення коефіцієнта поділу попереднього подільника.

Мікроконтролери сімейства PIC16C5X, крім PIC16C52, мають вбудований сторожовий таймер WDT, який може бути вимкнений лише через біт конфігурації, що задається при програмуванні. Для підвищення надійності він працює від власного RC-генератора. Таймер скидання DRT призначений для підтримування контролера в скинутому стані протягом перехідних процесів після ввімкнення живлення для стабілізації роботи генератора. Наявність цих таймерів дозволяє в багатьох випадках відмовитися від схеми зовнішнього скидання.

Режим зниженого енергоспоживання SLEEP призначений для забезпечення дуже низького струму споживання в режимі очікування (менше 1 мкА при вимкненому сторожовому таймері). Вихід з режиму SLEEP можливий за зовнішнім сигналом чи скиданням після закінчення відліку сторожового таймера. Можливість вибору типу генератора дозволяє ефективно використовувати мікроконтролери сімейства для різних застосувань. Використання RC-генератора дозволяє зменшити вартість системи, а LP-генератор на низькочастотному кварцовому резонаторі знижує енергоспоживання. Захист коду програми й установка коду ідентифікації здійснюється установленням відповідних розрядів у слові конфігурації при програмуванні.

Для розроблення й налагодження програмних модулів вільно поширюється асемблер MPASM, симулятор MPSIM та інтегрована система налагодження MPLAB.

Для апаратного підтримання режиму програмування мікросхем є різні типи програматорів, які під'єднуються до комп'ютерів типу IBM PC. Існують типи програматорів, які можна під'єднати безпосередньо до робочої плати контролера, що значно прискорює налагодження. Програмування мікро-

контролерів виконується через послідовний канал, у якості якого використовують два розряди порту вводу-виводу. Режим програмування задають шляхом встановлення на виводі скидання мікроконтролера напруги +12 В.

1. Контролер PIC16C84. Організація, програмування та застосування

Наступне покоління контролерів PIC16 (відносно PIC16C54) утворює сімейство PIC16C8х. Контролер PIC16C84 має внутрішню пам'ять програм (EEPROM) об'ємом 1 к×14 бітів та 64 байти EEPROM пам'яті даних.

Усі команди мають довжину в одне слово шириною у 14 бітів та виконуються за один цикл, крім команд переходу, які виконуються за два цикли. На максимальній тактовій частоті 10 МГц один цикл триває 400 нс. PIC16C84 підтримує переривання від чотирьох джерел і восьмирівневий апаратний стек.

Периферія контролера містить: 8-бітний лічильник-таймер з 8-бітним програмованим попереднім подільником, 13 ліній двонаправленого вводу-виводу. Лінії вводу-виводу мають високу навантажувальну здатність. Лінія може передати навантаженню максимальний струм 25 мА, а прийняти – 20 мА. що спрощує зовнішні драйвери, та зменшує загальну вартість системи.

Серія PIC16C84 підходить для широкого спектра схем від високошвидкісного керування автомобільними й електричними двигунами до економічних віддалених прийомопередавачів, вимірювальних приладів та телекомунікаційних процесорів. Наявність EEPROM даних дозволяє розміщувати параметри прикладних програм (коди передавачів, швидкості двигуна, частоти, тощо) у енергонезалежній пам'яті.

Малі розміри корпусів, як для звичайного, так і для поверхневого монтажу, робить цю серію мікроконтролерів придатною для портативних систем. Низька вартість, економічність, швидкодія, простота використання і гнучкість вводу-виводу робить PIC16C84 привабливим навіть у областях, де раніше не застосовувалися мікроконтролери.

Вбудований автомат програмування EEPROM кристала PIC16C84 дозволяє легко підлаштовувати програму і дані під конкретні вимоги навіть після завершення виготовлення плати. Ця можливість може бути використана як для тиражування, так і для занесення каліброваних даних після остаточного тестування.

Система команд контролера складають лише 35 команд, котрі виконуються за один цикл (крім команд переходу). Кожна команда займає у пам'яті 14 біт та призначена для виконання дій над 8-бітними даними з використанням прямої, безпосередньої, непрямой та відносної адресації даних та команд.

Статичний принцип в архітектурі та економічна високошвидкісна КМОН технологія виготовлення призводять до широкого діапазону робочої частоти при невеликому струмі споживання. Робоча тактова частота мікроконтролера може лежати у межах від 0 Гц до 10 МГц. Завдяки КМОН технології виготовлення у мікроконтролерах серії PIC16C8X досягнутий широкий діапазон напруги живлення, котра може становити від 2 до 6 В. Струм споживання становить 3 мА при живленні від 5 В та тактовій частоті 4 МГц, 50 мкА для живлення від 2 В при тактовій частоті 32 кГц та 26 мкА у режимі чекання при 2 В. Вмикання режиму економії енергії (SLEEP) суттєво знижує енергоспоживання. Одночасно досягнута можливість отримання струму в 20÷25 мА з виходів мікроконтролера. Діапазон робочих температур у комерційному виконанні становить від 0 до +70 °C, у промисловому – від -40 до +70 °C, у автомобільному – від -40 до до +125 °C.

Мікроконтролери мають 12 ліній вводу-виводу з індивідуальним налаштуванням та можливістю генерації переривань за зміною стану, Watchdog-таймер (WDT) із власним убудованим генератором, що забезпечує підвищену надійність, 8-бітний лічильник-таймер RTCC з 8-бітним програмованим попереднім подільником, таймер ввімкнення при скиданні. Спеціальна схема у мікроконтролері дозволяє автоматичне скидання при ввімкненні живлення мікросхеми без використання додаткових елементів.

Вбудований генератор має біти для вибору конфігурації для роботи у режимі: RC-генератора (RC), звичайного кварцового резонатора (XT), високочастотного кварцового резонатора (HS), економічного низькочастотного резонатора (LP).

Пам'ять мікроконтролера поділяється на пам'ять даних, ПЗП програм, 8-рівневий апаратний стек та EPROM пам'ять даних. Також у мікроконтролерах виділяють 15 спеціальних апаратних регістрів SFR. EPROM має біт таємності для захисту коду, біти ідентифікації, тощо. У OMEOM є вбудований пристрій для програмування пам'яті програм, що дозволяє використовувати при програмуванні лише кілька ліній для програмування. У режимі програмування лінія RB6 використовується для подання тактової частоти, а лінія RB7 – для введення та виведення даних. Третя лінія MCLR/Vpp указує на режим програмування.

Усю пам'ять даних поділяють на дві частини: спеціальні регістри та ОЗП даних. Всі спеціальні регістри та лічильник команд відображаються на пам'ять даних. До складу спеціальних регістрів входять: регістр лічильника-таймера реального часу (TMR0), лічильник команд (PC), регістр стану (STATUS), регістри вводу-виводу (PORT) та регістр непрямої адресації (FSR). Крім того, спеціальні регістри керують конфігурацією портів вводу-виводу і режимів попереднього подільника. Регістри загального призначення використовуються програмою для збереження змінних. Адресація всіх регістрів і комірок пам'яті у мікроконтролерах сімейства PIC16C8X може бути прямою та непрямою.

У мікроконтролерах PIC16C8X використовують 4 джерела переривання: зовнішній вхід INT, переповнення таймера, переривання за зміною сигналів на лініях порту В та переривання за завершенням записування даних у пам'ять EEPROM.

Мікроконтролери PIC16C8X теж мають ортогональну (симетричну) систему команд, що дозволяє виконувати будь-яку операцію з будь-яким регістром, використовуючи будь-який метод адресації. Це полегшує програмування для них і значно зменшує час, необхідний на навчання роботи з

ними.

1.1. Відмінності PIC16C84 від PIC16C5х

У PIC16C84 довжина команди збільшилася до 14 бітів. Це дозволило організувати ПЗП і ОЗП з більшим розміром сторінки. Сторінка ПЗП має тепер розмір 2 кбайти замість 512 байтів у PIC16C5х, сторінка ОЗП – 128 байтів замість 32.

Біти вибору сторінок RA2, RA1, RA0, що використовувалися у кристалі PIC16C5X, вилучено з регістра STATUS. Перемиканням сторінок у програмній пам'яті керує регістр програмного лічильника (PCLATH).

Додано чотири нові команди: RETURN, RETFIE, ADDLW, SUBLW. Зникла необхідність у двох командах TRIS і OPTION, однак їх позначення збережені для програмної сумісності з PIC16C5X. Регістри OPTION і TRIS виконано адресованими за номером.

Додана можливість працювати за перериванням за вектором 0004h, а стек збільшено до восьми рівнів, що значно полегшує написання програм. Адреса скидання при ввімкненні змінилася на 0000h. Розпізнаються п'ять різних типів скидань (виходів з режиму SLEEP). Змінено ініціалізацію регістрів, вони встановлюються залежно від типу скидання. Додано вихід з режиму SLEEP через переривання.

Для надійнішого запуску додано апаратні затримки від таймера запуску (OST) та таймера ввімкнення живлення (PWRT). Ці таймери можуть бути використані вибірково, аби уникнути небажаних очікувань при ввімкненні та при виході з режиму SLEEP.

Порт В має активні навантаження і переривання при зміні вхідних сигналів. Регістр з адресою 07h (порт C) у контролерах PIC16C84 не існує і не є регістром загального призначення, а вивід RTCC об'єднаний з виводом порту (RA4).

Регістр FSR (F4), в якому зберігається вказівник при непрямій адресації, став шириною вісім бітів. Реалізовано вбудований автомат програмування

EEPROM.

1.2. Перехід від PIC16C5х до PIC16C84

Щоб перевести код, написаний для PIC16C5X, у код для PIC16C84, необхідно виконати такі дії:

- ◇ Замінити операції вибору сторінок програмної пам'яті (керування бітами RA2, RA1, RA0) командами CALL чи GOTO.
- ◇ Переглянути всі адреси переходів, які обчислюються у програмі та впевнитися, що біти сторінок встановлюються коректно для нового кристала.
- ◇ Вимкнути перемикання сторінок пам'яті даних. Перевизначити змінні й перерозподілити пам'ять для них.
- ◇ Перевірити формат даних у регістрах STATUS, OPTION, FSR.
- ◇ Змінити вектор скидання при ввімкненні на 0000h.
- ◇ Видалити звертання до комірки за адресою 07h пам'яті даних.

2. Умове позначення та призначення виводів мікросхеми

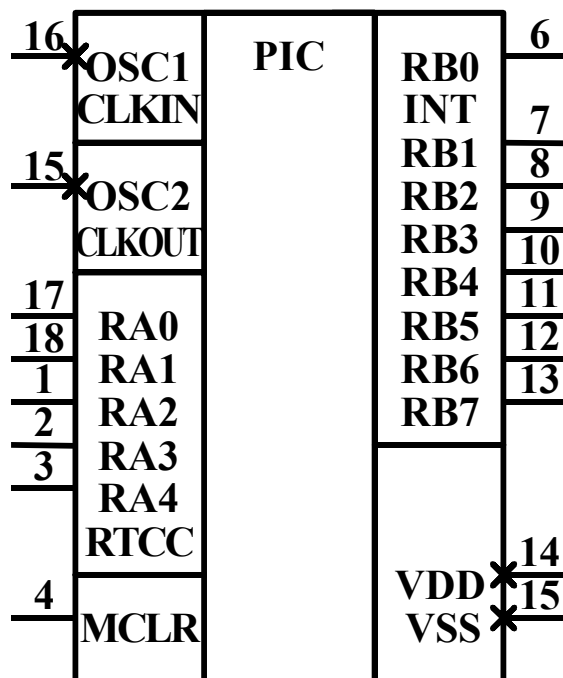


Рис. 76. Умове позначення мікросхеми

Умове позначення мікросхеми зображено на рис. 76. Позначення виводів, та їх призначення наведені у таблиці 45.

Таблиця 45. Призначення виводів мікросхеми

Позначення	Номер виводу	Призначення
RA0-RA3	17, 18, 1, 2	Двонаправлені лінії вводу-виводу. Вхідні та вихідні рівні відповідають TTL
RA4 RTCC	3	Двонаправлена лінія вводу-виводу. Вхід через тригер Шмітта. Вивід працює як вивід з відкритим стоком, також може використовуватись як вхід частоти для лічильника-таймера. В такому режимі роботи вивід має тригер Шмітта на вході
RB0/INT	6	Двонаправлена лінія вводу-виводу порту В. Лінія може використовуватись як вхід переривання, при цьому вхід буферизується тригером Шмітта
RB1 ÷ RB7	7÷13	Двонаправлені лінії вводу-виводу порту В. Лінії можуть бути запрограмовані у режимі внутрішніх активних навантажень на лінію живлення по усіх виводах. Виводи RB4...RB7 можуть бути програмно налаштовані як входи переривання по зміні стану на кожному із входів. При програмуванні МК RB6 використовується як тактовий, а RB7 як вхід/вихід даних. У режимі програмування лінії RB6 та RB7 мають на вході тригер Шмітта. В інших режимах сигнали на лініях відповідають КМОП TTL логіці
MCLR	4	Сигнал скиду. Логічний нуль на лінії призводить до скидання мікросхеми. В режимі програмування призначений для подавання напруги програмування. Лінія на вході має тригер Шмітта
OSC1 CLKIN	16	Вхід під'єднання кварцового резонатора, зовнішньої RC-ланки або зовнішнього джерела синхронізації. Вхід має тригер Шмітта, коли конфігурується в режимі RC-генератора і КМОП в усіх інших випадках
OSC2 CLKOUT	15	Вхід під'єднання кварцового резонатора, вихід сигналу синхронізації в режимі синхронізації за допомогою RC ланки, або в режимі включення зовнішнього тактового сигналу. Частота сигналу у режимі роботи з RC-ланкою дорівнює 1/4 частоти OSC1 і вказує швидкість виконання команд
VDD	14	Напруга живлення
VSS	5	Загальний (земля)

3. Структурна схема PIC16C84

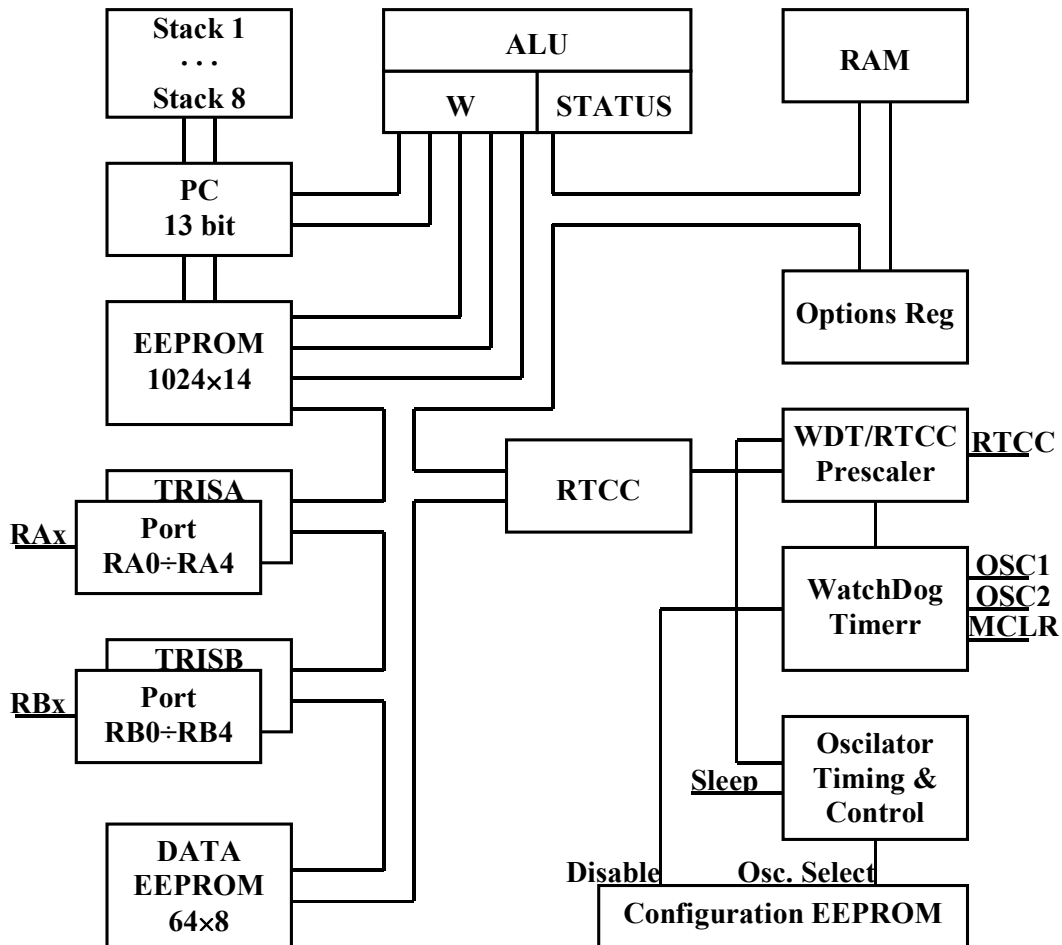


Рис. 77. Структурна схема мікроконтролера PIC16C84

Архітектура PIC16C84 (рис. 77) заснована на концепції роздільних шин і областей пам'яті для даних та команд, що носить назву гарвардської архітектури. Шина даних (Data Bus) і ОЗП даних (RAM) мають розрядність 8 бітів. Шина кодів команд та ПЗП програм (EEPROM) є 14-бітні. Така концепція забезпечує просту, але потужну систему команд, розроблену так, що бітові, байтові і реєстрові операції виконуються з високою швидкістю і з перекриттям з часом вибірок команд і циклів виконання. 14-бітна ширина програмної пам'яті забезпечує вибірку 14-бітової команди в один цикл.

Двоступінчатий конвеєр забезпечує одночасну вибірку і виконання команди. Усі команди виконуються за один цикл, крім команд переходів. У PIC16C84 програмна пам'ять розміром 1 к×14 розташована всередині кристала. Програма, що виконується, може знаходитися тільки в убудованому ПЗП.

4. Регістр STATUS

D7	D6	D5	D4	D3	D2	D1	D0
IPR	RP1	RP0	T0	PD	Z	DC	C

Рис. 78. Формат регістра STATUS

Регістр ознак у мікроконтролері PIC16C84 має назву STATUS. Він зберігає ознаки результатів арифметичних операцій, а також ознаки стана контролера. Формат регістра STATUS зображений на рис. 78.

Біт C є ознакою перенесення та позичання. Команда додавання встановлює цей біт у одиницю коли результат додавання перевищує число 255, та у нуль у протилежному випадку. Команда віднімання встановлює цей біт у нуль у випадку наявності позичання із дев'ятого розряду, а у одиницю при відсутності позичання. При виконанні команд зсуву цей біт завантажується з молодшого біта операнда при виконанні операції зсуву вправо, та з старшого при зсуві вліво.

Біт DC є ознакою десяткового перенесення або позичання. Команда додавання встановлює цей біт в одиницю, якщо було перенесення із молодшої тетради, та у нуль при відсутності перенесення. Команда віднімання встановлює біт у 1 у випадку відсутності позичання із четвертого розряду, а у нуль – при його наявності.

Біт Z є ознакою нульового результату. Він встановлюється, якщо результатом арифметичної чи логічної операції є нуль.

Біт PD (Power Down) вказує біт вмикання живлення. Біт призначений для вмикання режиму пониженого енергоспоживання. Біт PD може бути змінений деякими командами контролера: команда CLRWDT встановлює біт у 1, а команда SLEEP – скидає його.

Біт TO (Time Out) вказує на спрацювання Watchdog-сторожового таймера. Біт встановлюється у одиницю у момент вмикання живлення, за командами CLRWDT та SLEEP, скидання біта здійснюється лише у момент спрацювання Watchdog-таймера.

Стан бітів TO та PD регістра STATUS визначає тип початкової ініціалізації: вимкнення живлення, спрацювання watchdog-таймера, вихід із режиму зниженого енергоспоживання у результаті спрацювання watchdog-таймера або активізація зовнішнього сигналу MCLR. Стани сигналів TO та PD після кожної з подій, що змінює їх стан наведені у таблиці 46.

Таблиця 46. Зміна станів бітів TO і PD після подій, що їх змінює

TO	PD	Тип події
0	0	Вихід із SLEEP після спрацювання Watchdog-таймера
0	1	Спрацювання Watchdog-таймера у нормальному режимі
Незмінний	0	Вихід із SLEEP за зовнішнім сигналом MCLR
1	1	Ввімкнення живлення
Незмінний	Незмінний	Нульовий сигнал на лінії MCLR
1	1	Виконання команди CLRWDT
1	0	Виконання команди SLEEP

Таблиця 47. Сторінки та адреси при використанні прямої адресації

RP1	RP0	Сторінка	Адреса
0	0	0	00h–7Fh
0	1	1	80h–FFh
1	0	2	100h–17Fh
1	1	3	180h–1FFh

Біти RP1 та RP0 призначені для вибору сторінки пам'яті даних при прямій адресації. У таблиці 47 наведені номери сторінок, що використовують команди з прямою адресацією даних. На кожній сторінці розташований 128 байтів. У кристалі PIC16C84 реально використовується лише біт RP0, а біт RP1 може використовуватися у якості біта загального призначення.

Біт IRP призначений для вибору сторінки пам'яті даних при непрямій адресації. Нульове значення біта IRP0 задає звертання до сторінок 0 та 1, які об'єднуються між собою, одиничне – до сторінок 2 та 3. Цей біт у PIC16C84 не використовується за прямим призначенням, тому його теж можна використовувати в якості біта загального призначення.

Регістр ознак доступний для будь-якої команди. Однак, біти TO і PD встановлюються апаратно і не можуть бути записані у регістр звичайними командами. Варто відзначити, що команди арифметичних та логічних операцій,

спочатку зберігають результат виконання команди, а потім встановлюють біти у регістрі ознак на основі обчисленого значення. Наприклад, команда CLRf обнулить усі біти, крім бітів TO і PD, а потім встановить біт Z у одиницю, хоча після виконання цієї команди регістр STATUS не буде мати нульове значення. Тому варто рекомендувати для зміни регістра статусу використовувати лише команди встановлення та скидання бітів, що не змінюють біти статусу.

5. Структура пам'яті мікроконтролера

5.1. Організація вбудованого ПЗП

Програмний лічильник у PIC16C84 має розрядність 13 бітів і здатний адресувати 8192 слова програмної пам'яті, кожне довжиною 14 бітів. Однак фізично на кристалі є лише 1024 слова пам'яті програм з адресами 0h÷3FFh. Звертання за адресами вище 3FFh призводить до звертань до адресів у межах перших 1024 комірок.

Серед адресів комірок окрему функцію мають адреси 000h та 004h. Адреса 000h відповідає початку процедури скидання та початкової ініціалізації, а адреса 004h – процедурі оброблення переривання.

Щоб записати в програмну пам'ять, кристал має бути переведений у спеціальний режим, за якого на лінію MCLR подається напруга програмування V_{prg}, а напруга живлення V_{dd} повинна бути в межах 4,5...5,5 В. Записування у програмну пам'ять здійснюється послідовно, з використанням лише двох ліній. Зауважимо, що EEPROM PIC16C84 розрахована на обмежене число циклів стирання та записування.

5.2. Лічильник команд та стек

Зазвичай адресація пам'яті програм здійснюється за допомогою лічильника команд. Розрядність програмного лічильника – 13 бітів. Молодший байт програмного лічильника (PCL) доступний для читання і записування. Він відображений на комірку ОЗП з адресою 02h. Старший байт програмного лічильника (PCN) не може бути безпосередньо прочитаний чи записаний. Старший байт програмного лічильника може бути записаний через регістр

PCLATCH, адреса якого 0Ah. Старші біти PCLATCH (PCH) завантажуються залежно від виконуваної команди. У випадку виконання команди CALL чи GOTO з PCLATCH використовують лише біти у позиціях 4 та 3 (рис. 79). У випадку прямого пересилання даних у регістр PCL одночасно з пересиланням даних здійснюється копіювання усіх бітів регістра з PCLATCH у PCH (рис. 80).

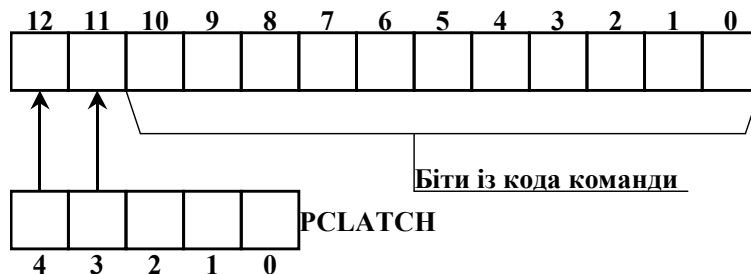


Рис. 79. Використання PCLATCH у формування адресів у випадку виконання команд GOTO та CALL

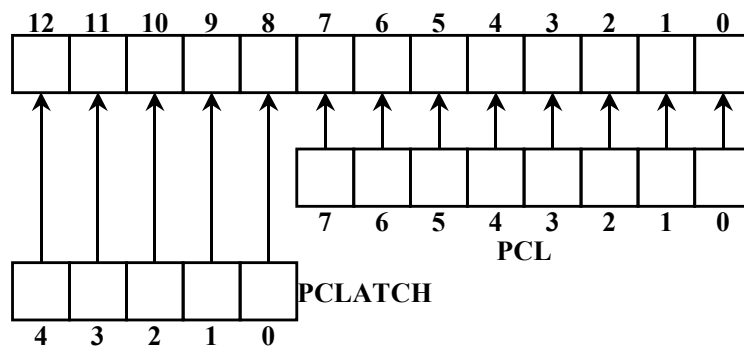


Рис. 80. Використання PCLATCH у формування адресів у випадку команд, що змінюють регістр PCL

У PIC16C84 найстарші два біти лічильника команд не використовують для адресації пам'яті програм, тому їх можна не встановлювати перед кожною командою CALL чи GOTO.

З лічильником команд пов'язаний апаратний стек, що використовується для збереження адресів переходів. PIC16C84 має восьмирівневий апаратний стек розрядністю 13 бітів. Область стека не належить ні до програмної області, ні до області даних, а покажчик стека користувачеві недоступний. Поточне значення програмного лічильника посилається в стек, коли виконується команда CALL або здійснюється опрацювання переривання. При виконанні процедури повернення з підпрограми (команди RETLW, RETFIE чи RETURN),

у програмний лічильник завантажується вміст стека. Регістр PCLATH не змінюється при операціях із стеком.

5.3. Структура ОЗП мікроконтролера

Адреса			Адреса	
00h	Непряма адреса ⁽¹⁾	Непряма адреса ⁽¹⁾	80h	
01h	TMR0	OPTION_REG	81h	
02h	PCL	PCL	82h	
03h	STATUS	STATUS	83h	
04h	FSR	FSR	84h	
05h	PORTA	TRISA	85h	
06h	PORTB	TRISB	86h	
07h	—	—	87h	
08h	EEDATA	EECON1	88h	
09h	EEADR	EECON2 ⁽¹⁾	89h	
0Ah	PCLATH	PCLATH	8Ah	
0Bh	INTCON	INTCON	8Bh	
0Ch	ОЗП даних банк 0 36 регістрів загального призначення	Відображається банк 0 адреси 0Ch-2Fh	8Ch	
2Fh				AFh
30h			B0h	
7Fh			FFh	
Банк 0		Банк 1		

- Фізично відсутні комірки, читаються як 0.

⁽¹⁾ - Регістр даних для непрямої адресації.

Рис. 81. Структура регістрів та ОЗП мікроконтролера

Структура ОЗП зображена на рис. 81. Область ОЗП організована у вигляді набору сторінок по 128 комірок, кожна комірка складається із 8 бітів. У область ОЗП відображені також усі регістри мікроконтролера. До комірок ОЗП

можна звертатися за прямою або непрямою адресами, а також через регістр вказівник FSR (адреса 04h).

У регістрі STATUS (адреса 03h) є біти вибору сторінок, що дозволяють звертатися до чотирьох сторінок старших модифікацій цього кристала. Однак для PIC16C84 пам'ять даних існує лише до адреси 02Fh, а перші 12 адресів кожної сторінки використовуються для розміщення регістрів спеціального призначення. Деякі регістри спеціального призначення продубльовані на обох сторінках, а інші розташовані лише на одній з двох існуючих сторінок (на першій або на нульовій).

За адресами 0Ch÷2Fh знаходяться комірки ОЗП для збереження будь-яких даних. У зв'язку із невеликим об'ємом ОЗП, комірки пам'яті також відображені на адреси 8Ch÷0AFh, котрі відносять до першої сторінки ОЗП.

Загальний адресний простір ОЗП складає 512 байтів. Комірки пам'яті можна адресувати за допомогою прямої та непрямої адресації. У випадку прямої адресації, молодші 7 бітів беруться з коду операції, а два старших біти – з регістра STATUS (рис. 82).



Рис. 82. Пряма адресація

У випадку непрямої адресації лише один біт береться із додаткового регістра, а решта бітів – з регістра-вказівника. Будь-яка команда, що використовує регістр з адресою 00h, звертається до комірки пам'яті за адресою, що зберігається у регістрі FSR (адреса 04h). Старший біт адреси визначається бітом IRP регістра STATUS. Читання з використанням непрямої адресації регістра з адресою 0 дає результат 00h. Запис у регістр з адресою 00h призведе до відкидання даних, що записуються.

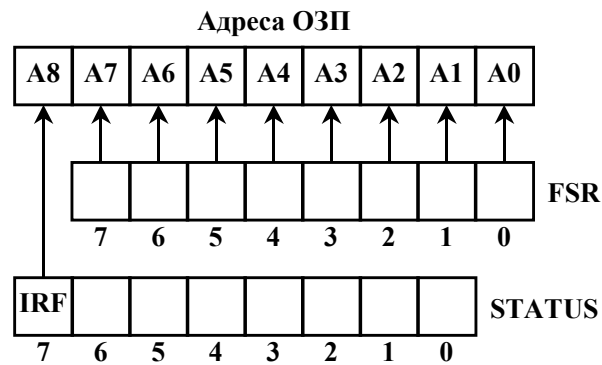


Рис. 83. Непряма адресація

Необхідна 9-бітна адреса формується об'єднанням умісту 8-бітного FSR регістра і біта IRP з регістра STATUS.

5.4. EEPROM даних

Уміст комірок EEPROM пам'яті даних зберігається при вимиканні живлення. Пам'ять даних EEPROM дозволяє читати та записувати інформацію окремими байтами. При записуванні байта автоматично стирається попереднє значення і записуються нові дані. Усі операції здійснює вбудований автомат EEPROM.

PIC16C84 має EEPROM пам'ять даних об'ємом 64×8 бітів. Читання та записування даних можливе під час роботи у всьому діапазоні напруги живлення. EEPROM пам'ять даних не відображена на адреси ОЗП. Доступ до неї здійснюється через два регістри: EEDATA (адреса 08h) та EEADR (адреса 09h). Регістр EEADR зберігає адресу комірки, до якої відбувається звертання. Регістр EEDATA використовується для доступу до даних зчитаних чи записуваних у EEPROM за адресою, котра записана у регістр EEADR. Додатково для визначення режиму роботи використовують два керуючих регістри: EECON1 (адреса 88h) та EECON2 (адреса 89h).

Регістр EECON2 (адреса 89h) використовують для запобігання помилкового записування даних у EEPROM. Записування даних починається лише за умови подання вірної ключової послідовності у регістр EECON2.

Регістр EECON1 (адреса 88h) – це керуючий регістр розрядністю п'ять бітів. Молодші п'ять бітів фізично існують, а старші три читаються завжди як 0.

D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	EEIF	WRERR	WREN	WR	RD

Рис. 84. Формат регістра EECON1.

Формат регістра EECON1 зображений на рис. 84. Керуючий біт RD (біт читання) встановлений в 1 запускає читання пам'яті даних EEPROM. Читання займає один цикл. Біт може бути лише встановлений. Біт скидається автоматично після завершення операцій читання.

Біт WR запускає записування. Біт може бути лише встановлений програмною в 1. Біт скидається автоматично після завершення операції записування. Встановлення біта у стан 1 вимагає попереднього подання ключової послідовності у регістр EECON2.

Біт WREN дозволяє записування у EEPROM пам'ять даних. Встановлення біта WREN у 1 дозволяє записування даних, встановлення у 0 – забороняє його. Після ввімкнення живлення біт WREN обнулюється.

Біт WRERR є ознакою помилки записування даних. Ознака помилки WRERR встановлюється в 1, коли процес записування переривається сигналом скидання MCLR або сигналом скидання від WDT таймера. Рекомендується перевіряти ознаку WRERR і за необхідності повторити процес записування даних. Незалежно від стану біта WRERR дані та адреса зберігаються у регістрах EEDATA і EEADR.

Ознака EEIF встановлюється в одиницю, коли вбудований автомат завершує записування в пам'ять даних. Біт повинен скидатись програмно. Біт пов'язаний із бітом дозволу переривання – EEIE в регістрі INTCON.

Після скидання при ввімкненні регістр набуває стан 0000x000.

При зчитуванні даних з пам'яті EEPROM потрібно записати необхідну адресу в EEADR регістр і потім встановити біт RD в одиницю. Дані з'являться в наступному командному циклі в регістрі EEDATA і можуть бути прочитані. Дані в регістрі EEDATA зафіксуються до наступного зчитування з EEPROM або програмної зміни даних у самому регістрі.

При записуванні в пам'ять EEPROM потрібно спочатку записати

необхідну адресу в EEADR регістр і дані в EEDATA регістр. Потім виконати спеціальну послідовність команд, що запускає безпосереднє записування:

```
movlv 55h      ;  
movwf EECON2   ;  
movlv AAh      ;  
movwf EECON2   ;  
bsf EECON1,WR; встановити біт WR, почати записування.
```

Під час виконання цієї частини програми всі переривання повинні бути заборонені для точного виконання часової діаграми. Час записування – приблизно 10 мс. Фактичний час записування буде змінюватися залежно від напруги, температури й індивідуальних властивостей мікросхеми. У кінці записування біт WR автоматично обнулюється, а ознака завершення записування EEIF, він же запит на переривання, встановлюється.

Для запобігання випадкових записів у пам'ять даних передбачено спеціальний біт WREN у регістрі EECON1. Рекомендується тримати біт WREN вимкненим, крім тих випадків, коли потрібно оновити пам'ять даних. Більш того, кодові сегменти, що встановлюють біт WREN і ті, котрі записують дані, повинні зберігатися за різними адресами, аби уникнути випадкового виконання їх обох при збої програми.

6. Порти

PIC16C84 має два порти: 5-бітний порт RA та 8-бітний порт RB з побітним індивідуальним настроюванням на ввід чи на вивід. Порт A – це порт розрядністю 5 бітів, якому відповідають лінії RA4÷RA0. Лінії RA3÷RA0 двонаправлені, а лінія RA4 – вихід з відкритим колектором. Адреса регістра порту A дорівнює 05h. Віднесений до порту A керуючий регістр TRISA розташований на першій сторінці регістрів за адресою 85h. TRISA4÷TRISA0 – це регістр розрядністю 5 бітів. Якщо біт керуючого регістра TRISA має одиничне значення, то відповідна лінія буде встановлюватися на ввід. Нуль перемикає лінію на вивід і одночасно виводить на неї вміст відповідного

регістра. На рис. 85 наведено схему портів RA0..RA3.

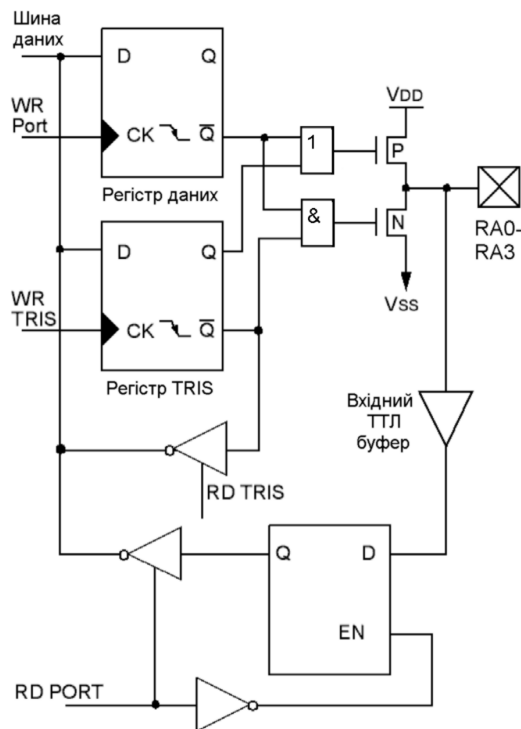


Рис. 85. Структура ліній RA0...RA3

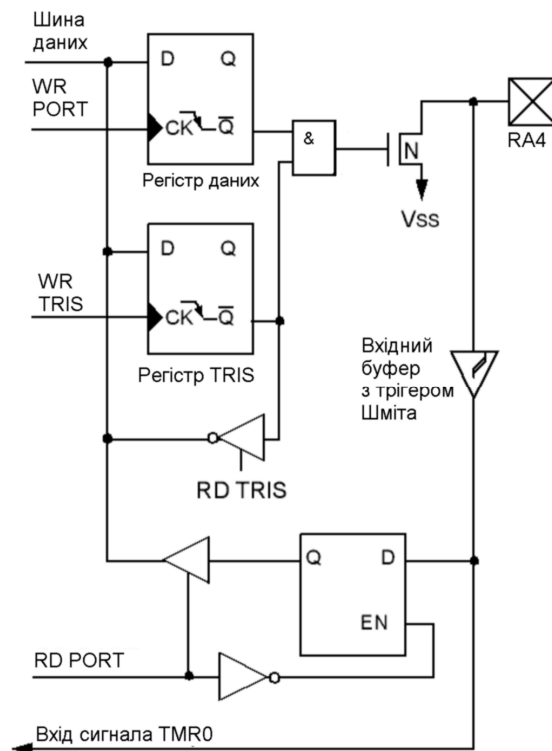


Рис. 86. Структура лінії RA4

Вивід RA4/RT має дещо іншу схему (рис. 86). Лінія RA4/RT використовується як вхід зовнішнього тактового сигналу для RTCC.

Порт В – це двонаправлений порт розрядністю у вісім бітів. Структура ліній RB3÷RB0 зображена на рис. 87, а ліній RB7÷RB4 – на рис. 88. Звертання до порта здійснюється за адресою 06h. Віднесений до порту В керуючий регістр TRISB розташований на першій сторінці регістрів за адресою 86h. Якщо біт керуючого TRISB регістра має значення, що дорівнює одиниці, то відповідна лінія буде встановлюватися на ввід. Нуль перемикає лінію на вивід і одночасно виводить на неї вміст відповідного регістра.

До кожної лінії порту В можна під'єднати невелике активне навантаження (підтяжку) з максимальним струмом до 100 мкА, що з'єднує вивід із лінією живлення. Навантаження автоматично від'єднується, якщо лінія запрограмована на вивід, а керуючий біт RBPU регістра OPTION (рис. 90) може від'єднати усі навантаження, встановленням біта RBPU у 1. Скидання при ввімкненні живлення також від'єднує всі навантаження.

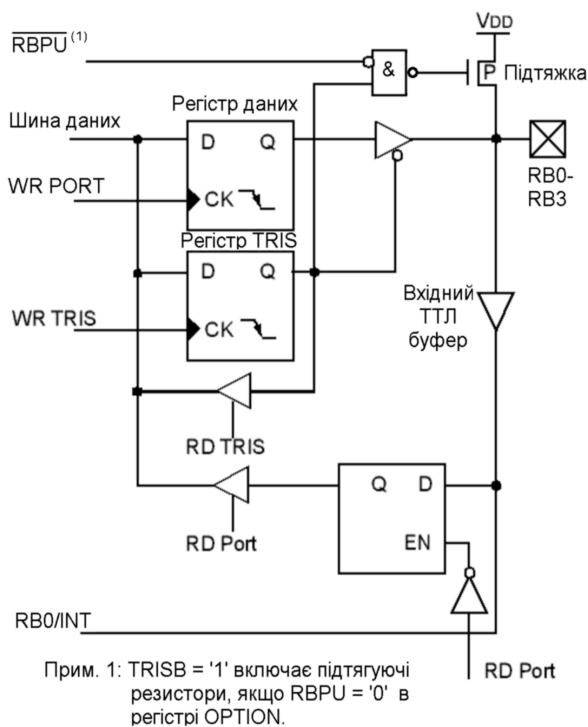


Рис. 87. Схема ліній RB3÷RB0

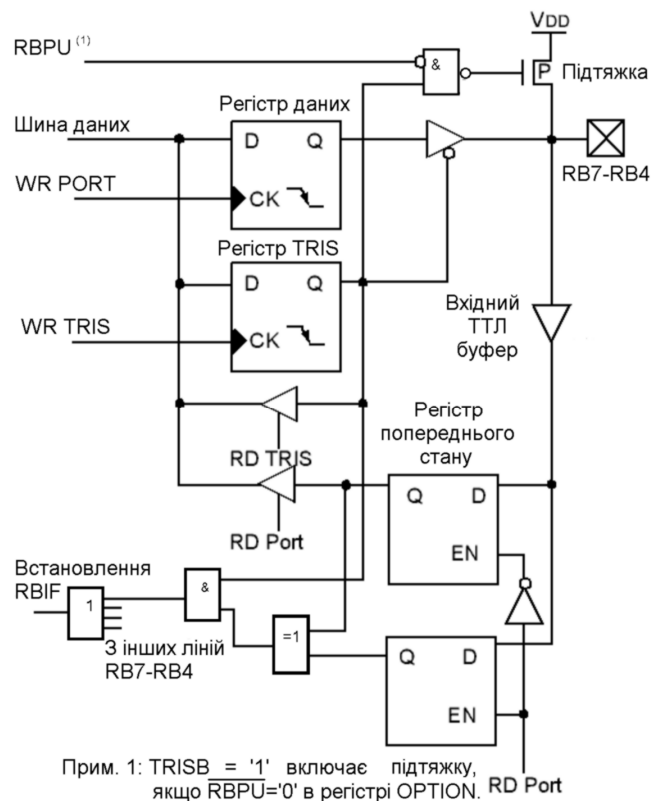


Рис. 88. Схема ліній RB7÷RB4

Чотири лінії порту В (RB7÷RB4) мають здатність викликати переривання при зміні значення сигналу на кожній з них. Якщо ці лінії запрограмовані на ввід, то вони опитуються в циклі читання. Нова величина вхідного сигналу порівнюється зі старою, збереженою в тригері, в кожному командному циклі. При розбіжності значення сигналу на виводі й у тригері генерується високий рівень. Виходи детекторів розбіжностей RB4, RB5, RB6, RB7 з'єднуються за схемою АБО та генерують переривання RBIF. Переривання може вивести кристал з режиму SLEEP. Зауважимо, що будь-яка лінія, запрограмована на вивід, не бере участі у порівнянні.

У підпрограмі опрацювання переривання потрібно скинути запит переривання одним із таких способів:

- ◇ заборонити переривання за допомогою обнулення біта RBIE;
- ◇ прочитати порт В. Це завершить процедуру порівняння;
- ◇ обнулити біт RBIF INTCON.

Переривання за розбіжністю і програмно встановлювані внутрішні активні навантаження на цих чотирьох лініях можуть забезпечити простий

інтерфейс, наприклад, із клавіатурою, з виходом із режиму SLEEP після натискання кнопок. Вивід RB0 з'єднаний із входом зовнішнього переривання INT.

Деякі команди виконують дії по алгоритму: читання – модифікація – записування даних. Наприклад, команди BCF і BSF зчитують порт повністю, модифікують один біт і виводять результат назад. Використання цих команд з портами вводу-виводу потребує обережності. Наприклад, команда BSF для біта 5 регістра PORTB спочатку зчитує всі вісім бітів. Потім виконує дії над бітом 5 і нове значення байта записується у вихідні тригери. Якщо інший біт регістра PORTB використовується в якості двонаправленої лінії введення-виведення даних (наприклад, біт 0) і в даний момент він визначений як вхідний, вхідний сигнал на цьому виводі буде зчитаний і записаний у вихідний тригер цього виводу, затираючи його попередній стан. Доки цей вивід залишається в режимі вводу, ніяких проблем не виникає. Але якщо пізніше лінія 0 перемкнеться в режим виводу, її стан буде визначеним не тим значенням, що було записано, а тим значенням, що було зчитано з порта у момент модифікації бітів командою BSF. Ще одна проблема може виникнути у випадку модифікації стану ліній перевантаженої лінії. Визначення стану порта, що працює на вивід даних здійснюється за станом виходу. У випадку тимчасового перевантаження лінії стан, що записаний у регістр виводу, може не відповідати стану лінії і буде втрачений. Для уникнення вищезгаданих проблем варто зберігати стан порта виводу у спеціальній змінній.

7. Таймери

7.1. Лічильник таймер RTCC

Структурна схема лічильника RTCC та його взаємодія з попереднім подільником та Watchdog-таймером зображена на рис. 89. На структурній схемі позначенням MUX зображений електронний перемикач, а позначенням SYNC – синхронізатор. Варто звернути увагу, що синхронізатор працює лише з лічильником, а попередній подільник використання синхронізатора не потребує. Це збільшує граничну частоту роботи попереднього подільника.

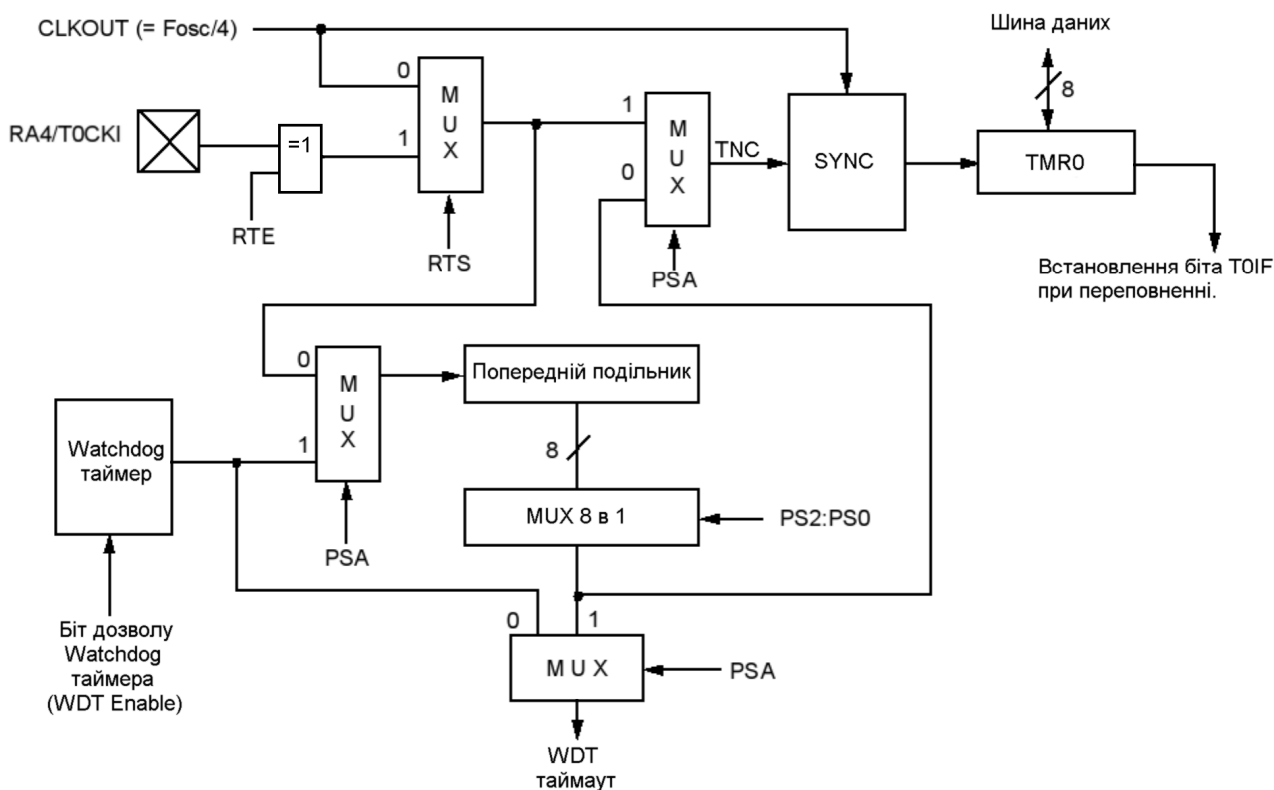


Рис. 89. Будова блока лічильників

D7	D6	D5	D4	D3	D2	D1	D0
RBP	INT EDG	RTS	RTE	PSA	PS2	PS1	PS0

Рис. 90. Формат регістра OPTION

Режими роботи таймера визначає регістр OPTION. Формат регістра зображений на рис. 90. Регістр OPTION (адреса 81h) доступний для читання і записування і містить різні керуючі біти, що визначають конфігурацію попереднього подільника, джерело сигналів лічильника-таймера, полярність сигналу INT та під'єднання активного навантаження на порту RB.

Режим таймера задають скиданням в нуль біта RTS у регістрі OPTION. У режимі таймера регістр TMR0 при відключеному попередньому подільнику буде інкрементуватися від внутрішнього джерела частоти кожного командного циклу.

Після записування інформації у TMR0, збільшення регістра розпочнеться після двох командних циклів. Це відбувається у випадку використання усіх команд, що здійснюють записування даних чи модифікацію регістра TMR0 (наприклад MOVF TMR0, CLRF TMR0). Для точнішого відліку часу необхідно

у лічильник-таймер завантажувати скоректоване значення. Якщо RTCC потрібно перевірити на рівність нулю без зупинки відліку, слід використовувати інструкцію `MOVF W, TMR0`.

Режим лічильника задає встановлення в одиницю біта RTS у регістрі `OPTION`. У цьому режимі RTCC буде інкрементуватися або позитивним, або негативним фронтом на лінії RA4/RTCC. Тип фронту визначає біт RTE у регістрі `OPTION`. При RTE рівному нулю буде обрано позитивний фронт (перехід з нуля в одиницю), а при одиничному значенні біта – негативний фронт (перехід із одиниці у нуль).

Попередній подільник може бути використаний разом із лічильником-таймером, чи з Watchdog-таймером. Варіант під'єднання подільника контролює біт PSA в регістрі `OPTION`. При `PSA = 0` подільник буде приєднаний до лічильника-таймера. Уміст подільника програмно недоступний. Коли подільник під'єднаний до лічильника-таймера, всі команди, які записують у лічильник скидають подільник у нуль. Це дозволяє отримати визначене значення витримки часу або кількості імпульсів на вході RA4.

Лічильник-таймер генерує переривання у момент переповнення при переході від 0FFh до 00h. Разом із генерацією переривання встановлюється біт запиту RTIF у регістрі `INTCON`. Переривання можна замаскувати бітом RTIE у регістрі `INTCON`. Біт запиту RTIF повинен бути скинутий програмно у процедурі переривання. Переривання від лічильника-таймера не може вивести процесор з режиму SLEEP внаслідок того, що таймер не функціонує в цьому режимі.

Для надійної роботи лічильника-таймера у режимі лічильника подій слід дотримуватись певних вимог до сигналу, що поступає на його вхід. Сигнал TNC (рис. 89) стробує внутрішній сигнал синхронізації двічі протягом кожного командного циклу. Тому сигнал TNC повинен зберігати високий і низький рівні мінімум на 20 нс довше ніж два періоди синхронізації.

У випадку використання з лічильником-таймером попереднього подільника на лінію TNC подається сигнал, поділений попереднім

подільником. У цьому випадку співвідношення між періодом вхідного сигналу (T) та періодом сигналу синхронізації (T_{OSC}) можна виразити так:

$$T \geq \frac{(4 T_{OSC} + 40 \text{ нс})}{N}$$

З іншого боку, для вірної роботи попереднього подільника, низький і високий рівень його входу повинні тривати не менш 10 нс.

Вихід подільника синхронізується внутрішнім сигналом тактової частоти, тому час фактичного інкремента лічильника-таймера відстає від фронту сигналу TNC на час від $3 T_{OSC}$ до $7 T_{OSC}$. Тому вимірювання інтервалу між подіями може біти виконано з точністю до $4 T_{OSC}$, що становить коло 400 нс при резонаторі на 10 МГц.

7.2. Watchdog-таймер

Watchdog-таймер є незалежним вбудованим таймером з RC-генератором, який не вимагає ніяких зовнішніх кіл. Він працює навіть тоді, коли основний генератор зупиняється, як це буває при виконанні команди SLEEP. Таймер виробляє сигнал скидання. Вироблення таких скидань може бути заборонено шляхом записування нуля у біт WDT в слові конфігурації.

Номінальна витримка WDT складає 18 мс (без використання попереднього подільника). Вона залежить від температури, напруги живлення, особливостей мікросхеми. Мінімальне значення витримки складає 7 мс, максимальне – 33 мс.

Якщо вимагаються більші витримки, то до WDT може бути під'єднаний вбудований подільник з коефіцієнтом поділу до 1:128, який програмується шляхом записування в регістр OPTION. Тоді можуть бути реалізовані затримки до 2,5 секунди.

Команди "CLRWDT" і "SLEEP" обнулюють WDT і подільник, якщо він під'єднаний до WDT, що запускає витримку часу спочатку і запобігає на деякий час виробленню сигналу скидання. Якщо сигнал скидання від WDT все ж відбувся, то одночасно обнулюється біт TO в регістрі STATUS.

7.3. Попередній подільник

Той же самий восьмибітний лічильник може бути ввімкнений або перед лічильником-таймером або після Watchdog-таймера. Схема ввімкнення попереднього подільника зображена на рис. 89. Біти PSA та PS0÷PS2 в регістрі OPTION визначають, з яким пристроєм працює подільник, і налаштовують коефіцієнт поділу. Значення коефіцієнта залежно від значень бітів PS2÷PS0 наведені у таблиці 48.

Таблиця 48. Настроювання попереднього подільника

PS2...PS0	Коефіцієнт поділу		PS2...PS0	Коефіцієнт поділу	
	RTCC	WDR		RTCC	WDR
0 0 0	1 : 2	1 : 1	1 0 0	1 : 32	1 : 16
0 0 1	1 : 4	1 : 2	1 0 1	1 : 64	1 : 32
0 1 0	1 : 8	1 : 4	1 1 0	1 : 128	1 : 64
0 1 1	1 : 16	1 : 8	1 1 1	1 : 256	1 : 128

Коли подільник під'єднаний до лічильника-таймера, всі команди, які записують у RTCC скидають подільник у нуль. Коли ж попередній подільник під'єднаний до Watchdog-таймера, то попередній подільник разом з Watchdog-таймером буде обнулюватися командою CLRWDT. У будь-якому випадку вміст подільника недоступний для програм.

Під'єднання попереднього подільника керується програмним шляхом. Нижче наведений фрагмент програми перемикання з RTCC на WDT.

```

CLRWDT                ; скинути Watchdog-таймер
BSF OPTION, 3         ; перемкнути попередній подільник
CLRWDT                ; скинути Watchdog-таймер та
                        ; попередній подільник

```

Переключення попереднього подільника з WDT на RTCC здійснюється наступною процедурою.

```

CLRWDT                ; скинути Watchdog-таймер
BSF OPTION, 3         ; перемкнути попередній подільник

```

Зазначеними програмами варто користуватися, навіть якщо WDT заборонений.

8. Переривання

В PIC16C84 переривання можуть бути від чотирьох джерел:

- ◇ зовнішнє переривання з лінії RB0/INT,
- ◇ переривання від переповнення лічильника-таймера RTCC,
- ◇ переривання після закінчення записування даних у EEPROM,
- ◇ переривання від зміни сигналів на ніжках порту RB7÷RB4.

Усі переривання мають однаковий вектор та адресу 0004h за якою знаходиться точка входження у процедуру переривання. Але в керуючому регістрі переривань INTCON записується, від якого саме джерела надійшов запит переривання. Формат регістра переривань INTCON зображений на рис. 91.

Біт загального дозволу переривань GIE у випадку встановлення в 1 дозволяє всі індивідуально немасковані переривання, встановлення біта у нуль забороняє всі переривання. Біт GIE обнулюється при скиданні.

Коли викликається переривання, біт GIE автоматично обнулюється, щоб заборонити подальші переривання, адреса повернення посилається в стек, а в програмний лічильник завантажується адреса 0004h. Час реакції на переривання для зовнішніх подій, таких, як переривання від лінії INT чи порту В, складає п'ять машиних циклів. Що на один цикл менше, ніж для внутрішніх подій, таких, як переривання за переповненням від таймера RTCC.

Команда повернення RETFIE завершує підпрограму обслуговування переривання і встановлює біт GIE для дозволу переривань.

D7	D6	D5	D4	D3	D2	D1	D0
GIE	EEIE	RTIE	INTE	RBIE	RTIF	INTF	RBIF

Рис. 91. Формат регістра INTCON

У підпрограмі опрацювання переривання джерело, що викликало переривання може бути визначене за відповідним бітом у регістрі INTCON. Єдиним винятком є переривання за кінцем записування в EEPROM. Його біт знаходиться у регістрі EECON1.

Одиничне значення біта RBIF у регістрі INTCON задає ознаку переривання від зміни стана ліній RB7÷RB4. Ознака встановлюється, коли сигнал на одному із входів RB7÷RB4 змінює свій рівень. Якщо лінія налаштована на вивід даних зміна її стана не викликає переривання. Ознака RBIF має бути скинута підпрограмою опрацювання переривання.

Переривання може бути заблоковане бітом RBIE. Одиничне значення біта дозволяє переривання від входів RB7÷RB4, а нульове – забороняє його.

Одиничне значення біта INTF указує на переривання від лінії RB0/INT. Ознака встановлюється, коли на лінії RB0/INT з'являється сигнал від зовнішнього джерела переривання. Полярність сигналу визначає біт INTEDG у регістрі OPTION. Якщо біт INTEDG має одиничне значення переривання викликається за переднім фронтом сигналу, якщо нульове – за заднім.

Переривання може бути замасковане скиданням керуючого біта INTE в нуль. Біт запиту INTF повинен бути очищений програмою переривання перед тим, як знову дозволити це переривання. Переривання INT може вивести процесор з режиму SLEEP, якщо перед входом у цей режим біт INTE був встановлений в одиницю.

Переповнення лічильника RTCC (перехід з стана 0FFh у стан 000h) встановить біт запиту RTIF у одиницю. Це переривання може бути дозволене встановленням біта RTIE у одиницю, скидання біта у 0 забороняє його. Так само як і попередні ознаки переривань запит RTIF має скидати підпрограма опрацювання переривання.

Ознака запиту переривання після закінчення записування в EEPROM, EEIF встановлюється у кінці записування даних у EEPROM. Це переривання може бути замасковане скиданням біта EEIE. На відміну від попередніх, ознака запиту EEIF зберігається у регістрі EECON1. Скидання запиту EEIF має виконувати програма опрацювання переривання.

Стан ознак запитів переривань не залежить від відповідних маскуючих бітів. Отже у процесі роботи програма може перевірити стан запитів й

замаскованих переривань.

Біт загального дозволу переривань GIE скидається автоматично за наступними обставинами:

- ◇ після ввімкнення живлення,
- ◇ за зовнішнім сигналом MCLR при нормальній роботі та у режимі SLEEP,
- ◇ після закінчення затримки таймера WDT при нормальній роботі та у режимі SLEEP.

У процесі скидання регістр INTCON приймає значення 0000 000х, де х позначає невідомий стан.

9. Система команд мікроконтролера

Кожна команда PIC16C84 – має 14-бітний код, розділений на код операції та поле операндів, що беруть участь у команді. У полі операндів зазвичай розміщений номер регістра або комірки пам'яті, константа, адреса переходу чи виклику підпрограми, тощо. Система команд PIC16C84 містить у собі команди, що виконують дії над байтами, бітами, константами та команди передавання керування.

При розгляді команд використаємо позначення наведені у таблиці 49.

Таблиця 49. Позначення типів операндів у командах

Позначення	Значення
R	Джерело даних – регістр чи комірка пам'яті даних, над вмістом котрої команда виконує дії
D	Приймач даних – місце де буде розташований результат виконання команди. D може приймати значення 0, 1, W, F. Якщо D дорівнює 0 чи W, результат виконання команди буде записаний у регістр W, якщо ж D рівний 1 чи F результат буде розміщений у регістр чи комірку пам'яті використану як джерело даних
B	B позначає номер біта, що бере участь у команді. B може бути лише константою
K	Константа (так званий "літерал") яку використовує команда

Більшість команд PIC16C84 виконує протягом одного машинного циклу. PIC16C84 виконує за два цикли лише команди передавання керування та

команди пропускання дій за умовою. Команди пропускання дій мікросхема виконує за два такти лише у випадку вірності зазначеної у команді умови. Один машиний цикл складає чотири періоди тактової частоти, тому для генератора з частотою 4 МГц час виконання більшості команд буде рівний 1 мкс.

9.1. Команди роботи з байтами

Команди роботи з байтами у документації фірми MicroChip носять назву байт-орієнтованих команд ("Byte Oriented Operations"). До них відносять команди пересилання даних, арифметичних та логічних операцій, зсувів, скидань. Усі байт-орієнтовані команди наведені у таблиці 50.

Таблиця 50. Байт-орієнтовані команди

Позначення	Назва команди	Ознаки
1	2	3
	Команди пересилання даних	
MOVF R, D	Пересилання регістра R у регістр заданий операндом D та порівняння R з нулем. Якщо D дорівнює W (0) вміст регістра R буде записаний у робочий регістр W, якщо D дорівнює F (1) дані будуть записані у регістр R	Z
MOVWF R	Пересилання значення збереженого у W у регістр R	—
MOVLW K	Пересилання константи (літерала) у регістр W	—
SWAPF R, D	Обмін місцями тетрад в R. Якщо D рівний W результат виконання команди буде записаний у регістр W, у противному випадку тетради будуть обмінені у R	—
OPTION	Завантаження W в регістр OPTION. Команда залишена з метою сумісності із PIC16C54, вона не рекомендована для використання у нових програмах	—
TRIS R	Завантаження TRIS регістра для відповідного порта з робочого регістра W	—
	Арифметичні команди	
INCF R, D	Інкремент регістра R. Результат виконання команди, якщо D рівний W результат виконання команди буде записаний у регістр W, у противному випадку тетради будуть обмінені у R	Z
INCFSZ R, D	Інкремент регістра R та пропуск наступної команди, якщо результат нульовий. Результат збільшення на одиницю записується у регістр заданий другим операндом. Якщо другий операнд W (0) дані будуть записані у регістр W, у противному випадку – у регістр R	—

1	2	3
DECF R, D	Декремент регістра R. Команда працює аналогічно команді INCF R, D, але виконує віднімання одиниці	Z
DECFSZ R, D	Інкремент регістра R та пропуск наступної команди, якщо результат нульовий. Команда працює аналогічно команді INCF R, D, але виконує віднімання одиниці	–
ADDWF R, D	Додавання W з R. Якщо другий операнд W чи 0 результат додавання записується у регістр W, якщо ж F чи 1 – у регістр R	C, Z, DC
ADDLW K	Додавання W константою K	C, Z, DC
SUBWF R, D	Віднімання регістра W від регістра D. Якщо другий операнд W чи 0 результат віднімання контролер записує у регістр W, якщо F чи 1 – у регістр R	C, Z, DC
SUBLW K	Віднімання регістра W від константи K	C, Z, DC
	Логічні операції	
CLRF R	Скидання регістра R. У регістр R записується нуль 0	Z
CLRF	Скидання регістра W. У регістр W записується нуль	Z
ANDWF R, D	Логічне І W з R. Якщо другий операнд W чи 0 результат записується у регістр W, якщо F чи 1 – у регістр R	Z
ANDLW K	Логічне І константи K та W. Результат записується у регістр W	Z
RLF R, d	Зсув R вліво через перенесення. Вміст регістра R зсувається вліво. Старший біт регістра записується у біт C, а попереднє значення біта C – у молодший розряд регістра R. Якщо другий операнд W чи 0 результат буде записаний у регістр W, якщо F чи 1 – у регістр R	C
RRF R, d	Зсув R у праву сторону через перенесення. Вміст регістра R зсувається вправо. Молодший біт регістра записується у біт C, а попереднє значення біта C – у старший розряд регістра R. Якщо другий операнд W чи 0 результат буде записаний у регістр W, якщо F чи 1 – у регістр R	C
COMF R, D	Інверсія регістра R. Якщо другий операнд W (0) результат буде записаний у регістр W, якщо F (1) – у регістр R	Z
IORWF R, F	Логічне АБО W та R. Якщо другий операнд W чи 0 результат буде записаний у регістр W, якщо F чи 1 – у регістр R	Z
IORLW K	Логічне АБО константи та W. Логічне АБО W та K. Результат записується у регістр W	Z
XORWF R, D	Виключне АБО W та R. Якщо другий операнд W (0) результат буде записаний у регістр W, якщо F (1) – у регістр R	Z
XORLW K	Виключне АБО константи K та W. Результат записується у регістр W	Z

9.2. Команди роботи з бітами

Команди роботи з бітами у документації фірми MicroChip носять назву біт-орієнтованих команд ("Bit Oriented Operations"). До них відносять команди встановлення, скидання та аналізування окремих бітів. Усі біт-орієнтовані команди наведені у таблиці 51.

Таблиця 51. Команди роботи з бітами

Позначення	Назва команди
BCF R, B	Скинути біт у регістрі R. Скидає у нуль біт з номером B у регістрі R
BSF R, D	Встановити біт у регістрі R. Встановлює у одиницю біт з номером B у регістрі R
BTFSC R, B	Протестувати біт на нуль. Якщо біт B у регістрі R дорівнює 0 наступна команда пропускається, якщо ж ні – наступна команда виконується
BTFSS R, B	Протестувати біт на одиницю. Якщо біт B у регістрі R дорівнює одинці наступна команда пропускається, якщо ж ні – наступна команда виконується

9.3. Команди переходів

Команди переходів використовують для зміни послідовності виконання команд у програмі. До цієї групи відносять команду безумовного переходу, виклику підпрограми, повернення з підпрограми та повернення з переривання.

Таблиця 52. Команди переходів

Позначення	Назва команди
GOTO K	Перехід за адресою. Команда завантажує 11 бітів із кода команди у лічильник команд. Старші два біта визначають четвертий та третій біти з PCLATCH
CALL K	Виклик підпрограми за адресою. Значення адреси наступної команди зберігається у стеку, потім команда завантажує 11 бітів із кода команди у лічильник команд. Старші два біта визначають четвертий та третій біти з PCLATCH.
RETURN	Повернення з підпрограми. У лічильник команд із вершини стека записується збережене значення.
RETLW K	Повернення з підпрограми з завантаженням константи K у регістр W. Дії команди аналогічні команді RETURN, але додатково у регістр W записується константа K
RETFIE	Повернення з переривання. Дії команди аналогічні команді RETURN, але додатково у одиницю встановлюється біт загального дозволу переривань GIE

Також до групи команд переходів можна віднести команди, котрі змінюють регістр PCL, що використовується у якості молодшого байта лічильника команд. У момент зміни значення регістра PCL у старші біти лічильника команд завантажується значення із регістра PCLATCH.

9.4. Спеціальні команди

До спеціальних команд відносять команди: "немає операції", переведення контролера у режим зниженого енергоспоживання та команду скидання watchdog-таймера. Група спеціальних команд наведена у таблиці 53.

Таблиця 53. Спеціальні команди

Позначення	Назва команди
NOP	Команда "немає операції". Команда не виконує ніяких дій
SLEEP	Команда переводить мікроконтролер у режим зниженого енергоспоживання
CLRWDT	Скидання watchdog-таймера. Watchdog-таймер має скидатись у процесі виконання програми, що підтверджує нормальну роботу мікроконтролера

10. Режим зниженого енергоспоживання

Вхід у режим SLEEP здійснюється командою SLEEP. За цією командою, якщо WDT дозволений, він скидається і починає відраховувати час, скидається також біт PD у регістрі STATUS, а біт TO встановлюється в 1. Після чого вимикається вбудований генератор та припиняється виконання програми та робота лічильника-таймера. Порти вводу-виводу зберігають стан, який вони мали до входу в режим SLEEP.

Для зниження споживаного струму в цьому режимі вихідні лінії повинні мати значення, щоб не протікав струм між кристалом і зовнішніми колами.

Вихід з режиму SLEEP здійснюється в результаті:

- ◇ зовнішнього скидання – імпульсу низького рівня на виводі MCLR;
- ◇ скидання при спрацюванні WDT (якщо він дозволений);
- ◇ переривання з виводу INT, або при зміні стану лінії порту B, або при завершенні записування даних в EEPROM. У зв'язку з тим, що лічильник-таймер у будь-якому режимі синхронізується тактовим

сигналом генератора вихід з режиму SLEEP за перериванням від лічильника-таймера неможливий.

При першій вказаній події відбувається скидання всього пристрою. Дві інші події допускають продовження виконання програми.

Розглянемо вихід за зовнішнім скиданням. Біт PD у регістрі STATUS, який встановлюється при ввімкненні живлення, але обнулюється командою SLEEP, може бути використаний для визначення стану процесора до "прокидання": знаходився процесор у режимі SLEEP, чи було просто вимкнене живлення (холодний старт). Біт TO дозволяє визначити, чим був викликаний вихід з режиму SLEEP: зовнішнім сигналом на лінії MCLR чи спрацюванням WDT. Скидання за сигналом MCLR призводить до втрати стану внутрішніх регістрів та лічильника команд, тому його не варто використовувати для нормального виходу із режиму SLEEP без особливої необхідності.

Інші методи виходу не змінюють вміст регістрів мікроконтролера та зберігають адресу повернення на команду, що йде за командою SLEEP. Для виходу з режиму SLEEP за допомогою переривань вони повинні бути дозволені встановленням відповідної маски в регістрі INTCON. При виході з режиму SLEEP буде виконуватися команда, що йде за командою SLEEP, якщо біт загального дозволу переривань GIE обнулений, у протилежному випадку керування буде передано в підпрограму обслуговування переривань зі збереженням у стеку адреси команди, наступної за командою SLEEP.

У випадку спрацювання WDT у режимі зниженого енергоспоживання відбувається запуск генератора, та перехід до команди, що йде за командою SLEEP.

11. Скидання та початкове встановлення

11.1. Джерела скидання

PIC16X8X має кілька джерел сигналу скидання:

- ◇ при ввімкненню живлення (Power-on Reset) POR;
- ◇ по зовнішньому сигналу MCLR;

- ◇ по сигналу Watchdog таймера (WDT Reset);
- ◇ за таймером ввімкнення живлення;
- ◇ за таймером запуску генератора.

Сигнал скидання з джерел скидання формується за допомогою схеми, зображеної на рис. 92. Таймер ввімкнення живлення (PWRT) дає фіксовану витримку часу в 72 мс (номінальне значення) при ввімкненні живлення. Таймер працює на внутрішньому RC-генераторі. Він дозволяє дочекатись стабілізації напруги живлення в процесі запуску контролера.

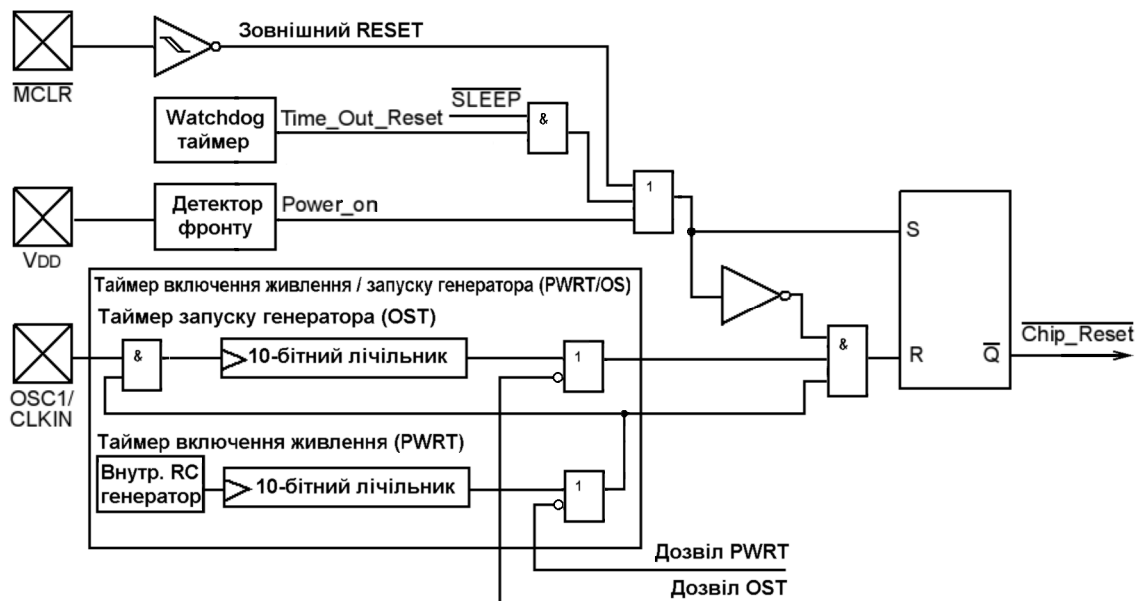


Рис. 92. Схема формування сигналу скидання

Таймер запуску генератора дозволяє після ввімкнення живлення або при виході із режиму зниженого енергоспоживання (SLEEP) дочекатися стабілізації частоти та режиму генерації кварцового чи керамічного резонатора. Таймер очікує надходження з генератора 1024 імпульсів перед зняттям сигналу скидання. Таймер ввімкнення живлення і таймер запуску генератора можуть бути вимкнені у разі необхідності. Таймер запуску генератора, найчастіше, не використовується при роботі з RC генератором, і при роботі з зовнішнім джерелом синхронізації.

Таймер ввімкнення живлення може не використовуватись у системах з зовнішнім формувачем сигналу скидання, наприклад, у системах з супервізором живлення.

11.2. Скидання при ввімкненні живлення

PIC16C84 має вбудований детектор ввімкнення живлення. Коли напруга живлення перевищить рівень 1,2...1,8 В, формується сигнал скидання, що включає таймер ввімкнення живлення запуску. Після закінчення витримки (близько 72 мс) вважається, що напруга досягла номіналу і запускається інший таймер витримки для стабілізації кварцового генератора. Програмований біт конфігурації дозволяє або забороняє витримку від згаданих вбудованих таймерів запуску. Витримка запуску змінюється залежно від кристала, живлення і температури.

Таймер для стабілізації генератора відраховує 1024 імпульси від генератора, який почав роботу. Вважається, що кварцовий генератор за цей час ввійшов у нормальний режим роботи. При використанні RC-генераторів витримка часу для стабілізації частоти не використовується.

Далі вмикається таймер чекання зовнішнього скидання MCLR. Це необхідно для тих випадків, коли потрібно синхронно запустити в роботу кілька PIC-контролерів через загальний для всіх сигнал MCLR. Якщо такого сигналу немає, то через час T_{ost} формується внутрішній сигнал скидання і контролер починає працювати за програмою. Час T_{ost} програмується бітами конфігурації в EEPROM.

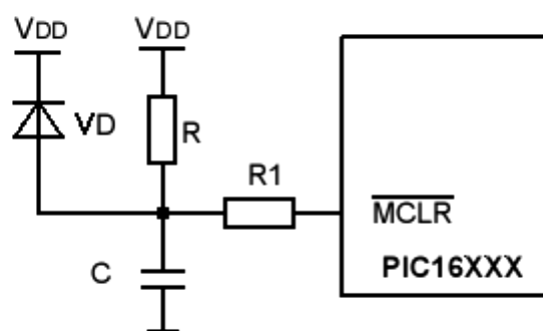


Рис. 93. Зовнішнє коло скидання

Коли VDD наростає надто повільно і, попри всі затримки на запуск, живлення ще не досягло свого мінімального значення VDD(min) для нормального функціонування, рекомендується використовувати зовнішні RC-ланки для скидання по сигналу MCLR (рис. 93). В інших випадках необхідності

в такому колі немає. Рекомендується вибирати опір резистора R не більше 40 кОм. Опір резистора R1, що обмежує струм через лінію вибирають у межах 100 Ом - 1 кОм.

11.3. Watcdog-таймер

Watchdog-таймер (WDT) є повністю незалежним таймером, що працює незалежно від інших блоків мікросхеми. Робота таймера полягає в очікуванні протягом певного періоду звертання від основної програми і формуванні сигналу скидання після закінчення заданого інтервалу часу у випадку відсутності звертань.

Команди CLRWDT та SLEEP обнулюють WDT і попередній подільник, якщо він під'єднаний до WDT. Якщо сигнал скидання від WDT все ж відбувся, то одночасно обнулюється біт TO в регістрі STATUS. Формування сигналу скидання може бути заборонено записом у біт конфігурації WDTE. Така операція виконується на етапі програмування мікросхеми.

Номінальна тривалість витримки часу WDT складає 18 мс (без використання попереднього подільника). Вона залежить від температури, напруги живлення та особливостей кристала. Якщо потрібна більший інтервал витримки часу, то до WDT може бути під'єднаний внутрішній попередній подільник з коефіцієнтом поділу до 128, що програмується шляхом запису в регістр OPTION. У результаті можуть бути реалізовані витримки часу до 2.5 секунди. Зауважимо, що такий метод формування затримок не є точним – параметри генератора WDT залежать від напруги живлення, температури та інших факторів.

12. Генератор та синхронізація

12.1. Типи генераторів

PIC16X8X може працювати з чотирма типами генераторів. Програміст може, користуючись двома конфігураційними бітами (FOSC1 і FOSC0), обрати один із режимів:

- ◇ RC-генератор.

- ◇ LP – низькочастотний кварцовий (керамічний) резонатор. Частоти резонансу до 100-200 кГц. Найчастіше використовують резонатори на 32 кГц та 200 кГц.
- ◇ XT – резонатор середніх частот. Частоти резонансу від 100-200 кГц до 4МГц.
- ◇ HS – високочастотний резонатор. Частоти резонансу від 4МГц і вище.

12.2. Робота з кварцовим резонатором

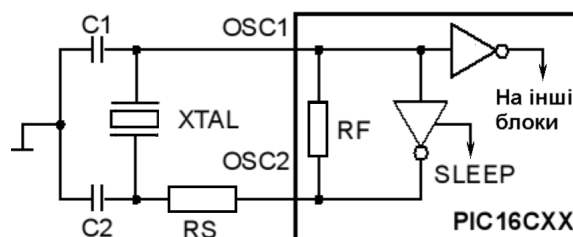


Рис. 94. Підключення кварцового резонатора

Схема включення кварцового резонатора показана на рис. 94. При роботі в режимах XT, LP, HS керамічний, або кварцовий резонатор під'єднується між виводами OSC1/CLKIN та OSC2/CLKOUT. Ємності C1 та C2 призначені для підвищення надійності запуску і роботи генератора: збільшення ємності призводить до підвищення стабільності, проте збільшує час запуску, який слід враховувати при виході з режиму зниженого енергоспоживання.

Типове значення ємності для резонаторів на частоту від 100 кГц і вище коло 33 пФ, для резонаторів на 32 кГц – 68 пФ. Резистор RS може бути потрібним для деяких типів резонаторів для гасіння вищих гармонік коливань.

Залежно від режиму роботи XT, LP, HS змінюється значення внутрішнього резистора RF, тому підбір типу резонатора необхідно здійснювати одночасно з вибором режиму роботи, користуючись при цьому документацією виробника мікросхеми.

12.3. Синхронізація від зовнішніх джерел

PIC16X8X може також синхронізуватися від зовнішніх джерел сигналу. Типова схема ввімкнення в такому режимі роботи зображена на рис. 95. При

роботі в цьому режимі обов'язково повинен бути обраний режим синхронізації від кварцового резонатора (XT, LP або HS), у випадку вибору RC-резонатора можливе пошкодження кристала.

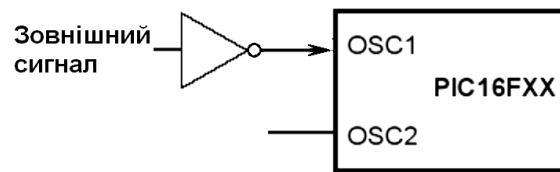


Рис. 95. Синхронізація від зовнішніх джерел

12.4. RC-генератор

Коли не пред'являються високі вимоги до точності відліку часу, зручно використовувати для тактування мікросхеми RC-генератор. Схема включення RC-генератора зображена на рис. 96. На виводі OSC2/CLKOUT видається сигнал з частотою, що дорівнює 1/4 частоти генератора. Сигнал може бути використаний для синхронізації інших мікросхем.

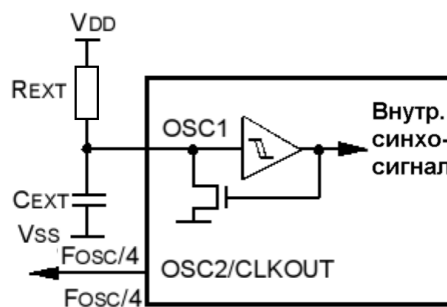


Рис. 96. RC генератор

Типові значення $R_{ext} = 5 - 100 \text{ кОм}$, $C_{ext} \geq 20 \text{ пФ}$. Наближені середні значення частоти синхронізації F_{osc} , похибка встановлення частоти при напрузі живлення 5В та температурі 25°C наведені в таблиці 54.

Таблиця 54. Середні значення частоти синхронізації

Cext	Rext	F _{osc}	Похибка	Cext	Rext	F _{osc}	Похибка	
20 пФ	3.3к	4.68 МГц	± 27%	100 пФ	10к	620 кГц	± 30%	
	5.1к	3.94 МГц	± 25%		100к	90.25 кГц	± 26%	
	10к	2.34 МГц	± 29%	300пФ		3.3 к	524 кГц	± 28%
	100к	250 кГц	± 33%		5.1к	415 кГц	± 30%	
100 пФ	3.3к	1.49 МГц	± 25%		100к	10к	270 кГц	± 26%
	5.1к	1.12 МГц	± 25%			100к	25.4 кГц	± 25%

Для значень R_{ext} , нижчих 2.2 кОм, генератор може працювати нестабільно або не запускатися взагалі. При дуже великих значеннях R_{ext} (наприклад, 1 МОм)

генератор стає чутливим до завад, вологості і струму через ізоляцію корпусу.

13. Конфігурація та захист

PIC16X8X має чотири слова, розташовані за адресами 2000H-2003H. Вони призначені для збереження ідентифікаційного коду (ID), контрольної суми або іншої інформації. Разом із конфігураційним словом, вони можуть бути прочитані та записані лише за допомогою програматора. Доступу за командами програми до них немає.

Якщо кристал захищений, користувачу рекомендується використовувати для ідентифікації тільки молодші сім бітів кожного слова ідентифікаційного коду, а в старші біти записувати 0.

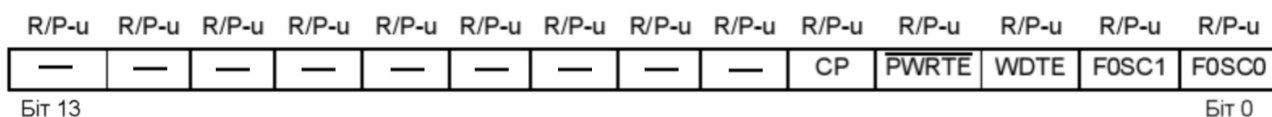


Рис. 97. Формат слова конфігурації

Призначення бітів слова конфігурації наведено в таблиці 55, а його формат – на рис. 97.

Таблиця 55. Призначення біт слова конфігурації

Позначення	Призначення.
CP	Біт захисту коду. Одиниця вказує, що захист коду ввімкнений, нуль – на відсутність захисту
PWRTE	Біт дозволу витримки часу після включення живлення. Одиниця вказує на наявність витримки в 72 мс при включенні, нуль – на відсутність витримки. Біт не впливає на роботу таймера запуску генератора
WDTE	Біт дозволу роботи Watchdog таймера. Одиниця вказує на дозвіл роботи Watchdog-таймера, нуль – на заборону скидання по Watchdog-таймеру
FOSC1, FOSC0	Біти вибору типу генератора. Усі комбінації крім FOSC1, FOSC0 = 11 указують на роботу з кварцовим або керамічним резонатором: FOSC1, FOSC0 = 00 задає режим з низькочастотним резонатором (LP); FOSC1, FOSC0 = 01 задає режим з середньочастотним резонатором (XT); FOSC1, FOSC0 = 01 задає режим з високочастотним резонатором (HS); FOSC1, FOSC0 = 01 задає роботу RC-генератором.

PIC16F84 має п'ять бітів конфігурації, що зберігаються в пам'яті програм і

встановлюються на етапі програмування кристала. Ці біти можуть бути запрограмовані (читаються як 0) або залишені незапрограмованими (читаються як 1) для вибору необхідного варіанта конфігурації мікросхеми. Вони розташовані в пам'яті програм за адресою 2007H. Слід зауважити, що ця адреса знаходиться поза областю кодів і програмно недоступна.

Програмний код, записаний на кристалі, може бути захищений від зчитування за допомогою встановлення біта захисту (CP) у слові конфігурації в нуль. Якщо встановлено захист, то біт CP можна стерти тільки разом зі стиранням умісту кристала.

Контрольні запитання

1. Основні технічні характеристики мікроконтролерів PIC16C5X.
2. PIC – контролер 16C84. Основні характеристики.
3. Архітектура PIC–контролера 16C84. Регістри контролера, їх призначення. Регістр статусу. RTCC таймер/ лічильник.
4. Архітектура PIC–контролера 16C84. Організація резидентної пам'яті даних та пам'яті програм. PC і адресація ПЗП.
5. Архітектура PIC–контролера 16C84. Система переривань. Зовнішнє переривання.
6. Переривання від RTCC. Переривання від порту RB. Переривання від EEPROM.
7. Архітектура PIC–контролера 16C84. Стек і повернення з підпрограм.
8. Архітектура PIC–контролера 16C84. Порти вводу/виводу. Регістри портів. Схема ліній порту A. Схема ліній порту B.
9. Правила записування програм мовою асемблера. Система команд PIC 16C84.
10. Правила записування програм мовою асемблера. Байт-орієнтовані команди.
11. Правила записування програм мовою асемблера. Операції з бітами.
12. Правила записування програм мовою асемблера. Переходи.

Список літератури

1. Сташин В. В., Урсов А. В., Мологонцева О. Ф. Проектирование цифровых устройств на однокристалльных микроконтролерах. /Сташин В. В., Урсов А. В., Мологонцева О. Ф. – М.: Энергоатомиздат, 1990. – 224 с.
2. Предко М. Руководство по микроконтролерам. /Предко М. – М.: Постмаркет, 2001–.
4. Зубчук В.И., Сигорский В.П., Шкуро А.Н. Справочник по цифровой схемотехнике. /Зубчук В.И., Сигорский В.П., Шкуро А.Н. – К.: Техніка, 1990.– 448 с.
5. Проектирование микропроцессорной электронно-вычислительной аппаратуры. Справочник. [Артюхов В. Г., Будняк А. А., Лапий В. Ю., Молявко С. М., Петренко А. И.] – К.: Техніка, 1988. – 263 с.
6. Микропроцессорный комплект К1810. Структура , программирование, применение./ [Казаринов Ю. М., Номоконов В. Н., Подклетнов Г. С., Филиппов Ф. В.] – М. Высшая школа., 1990. – 269 с.
7. Мікропроцесорна техніка. [Якименко Ю. І., Терещенко Т. О., Сокол Є. І., Жуйков В. Я., Петергера Ю. С.]; за ред. Терещенко Т. О., – К.: "Видавництво Політехніка", "Кондор" 2004. – 440 с.
8. Бродин В. Б., Калинин А. В. Системы на микроконтролерах и БИС программируемой логики./ Бродин В. Б., Калинин А. В. – М.: ЭКОМ, 2002. – 400 с.
9. Коффрон Дж. Технические средства микропроцессорных систем. Практический курс./ Коффрон Дж.; пер. с англ. Балыбердина А. В.– М. Мир,– 1983. – 344 с.
10. Однокристалльные микроЭВМ./ [Боборыкин А.В., Липовецкий Г.П., Литвинский Г.В., Оксинь О.Н., Прохорчик С.В., Проценко Л.В., Пертенко Н.В., Сергеев А.А., Сивобород. П.В.]. – М.: МИКАП, 1994. – 400 с.
11. Ульрих В.А. Микроконтролеры PIC16X7XX./ Ульрих В.А. – [2-е изд. перераб и доп.]. – СПб.: Наука и Техника, 2002. – 320 с.

12. Тавернье К. PIC-Микроконтролер. Практика применения./ Тавернье К. – М.: ДМКПресс, 2004. – 272 с.

13. PIC16F84 Data Sheet. 18-pin 8-Bit CMOS EEPROM Microcontroller. – Microchip Technology Inc. 1996. – 108 с. – Режим доступа: <http://ww1.microchip.com/downloads/en/DeviceDoc/30445c.pdf>.

14. PIC16F84A Data Sheet 18-pin Enhanced FLASH/EEPROM 8-bit Microcontroller. – Microchip Technology Inc. 2001. – 86 с. – Режим доступа: <http://ww1.microchip.com/downloads/en/DeviceDoc/35007b.pdf>.